



Sun Java System Message Queue 4.2 Administration Guide



Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95054
U.S.A.

Part No: 820-4916
June 2008

Copyright 2008 Sun Microsystems, Inc. 4150 Network Circle, Santa Clara, CA 95054 U.S.A. All rights reserved.

Sun Microsystems, Inc. has intellectual property rights relating to technology embodied in the product that is described in this document. In particular, and without limitation, these intellectual property rights may include one or more U.S. patents or pending patent applications in the U.S. and in other countries.

U.S. Government Rights – Commercial software. Government users are subject to the Sun Microsystems, Inc. standard license agreement and applicable provisions of the FAR and its supplements.

This distribution may include materials developed by third parties.

Parts of the product may be derived from Berkeley BSD systems, licensed from the University of California. UNIX is a registered trademark in the U.S. and other countries, exclusively licensed through X/Open Company, Ltd.

Sun, Sun Microsystems, the Sun logo, the Solaris logo, the Java Coffee Cup logo, docs.sun.com, Java, and Solaris are trademarks or registered trademarks of Sun Microsystems, Inc. or its subsidiaries in the U.S. and other countries. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. in the U.S. and other countries. Products bearing SPARC trademarks are based upon an architecture developed by Sun Microsystems, Inc.

The OPEN LOOK and SunTM Graphical User Interface was developed by Sun Microsystems, Inc. for its users and licensees. Sun acknowledges the pioneering efforts of Xerox in researching and developing the concept of visual or graphical user interfaces for the computer industry. Sun holds a non-exclusive license from Xerox to the Xerox Graphical User Interface, which license also covers Sun's licensees who implement OPEN LOOK GUIs and otherwise comply with Sun's written license agreements.

Products covered by and information contained in this publication are controlled by U.S. Export Control laws and may be subject to the export or import laws in other countries. Nuclear, missile, chemical or biological weapons or nuclear maritime end uses or end users, whether direct or indirect, are strictly prohibited. Export or reexport to countries subject to U.S. embargo or to entities identified on U.S. export exclusion lists, including, but not limited to, the denied persons and specially designated nationals lists is strictly prohibited.

DOCUMENTATION IS PROVIDED "AS IS" AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

Copyright 2008 Sun Microsystems, Inc. 4150 Network Circle, Santa Clara, CA 95054 U.S.A. Tous droits réservés.

Sun Microsystems, Inc. détient les droits de propriété intellectuelle relatifs à la technologie incorporée dans le produit qui est décrit dans ce document. En particulier, et ce sans limitation, ces droits de propriété intellectuelle peuvent inclure un ou plusieurs brevets américains ou des applications de brevet en attente aux Etats-Unis et dans d'autres pays.

Cette distribution peut comprendre des composants développés par des tierces personnes.

Certains composants de ce produit peuvent être dérivées du logiciel Berkeley BSD, licenciés par l'Université de Californie. UNIX est une marque déposée aux Etats-Unis et dans d'autres pays; elle est licenciée exclusivement par X/Open Company, Ltd.

Sun, Sun Microsystems, le logo Sun, le logo Solaris, le logo Java Coffee Cup, docs.sun.com, Java et Solaris sont des marques de fabrique ou des marques déposées de Sun Microsystems, Inc., ou ses filiales, aux Etats-Unis et dans d'autres pays. Toutes les marques SPARC sont utilisées sous licence et sont des marques de fabrique ou des marques déposées de SPARC International, Inc. aux Etats-Unis et dans d'autres pays. Les produits portant les marques SPARC sont basés sur une architecture développée par Sun Microsystems, Inc.

L'interface d'utilisation graphique OPEN LOOK et Sun a été développée par Sun Microsystems, Inc. pour ses utilisateurs et licenciés. Sun reconnaît les efforts de pionniers de Xerox pour la recherche et le développement du concept des interfaces d'utilisation visuelle ou graphique pour l'industrie de l'informatique. Sun détient une licence non exclusive de Xerox sur l'interface d'utilisation graphique Xerox, cette licence couvrant également les licenciés de Sun qui mettent en place l'interface d'utilisation graphique OPEN LOOK et qui, en outre, se conforment aux licences écrites de Sun.

Les produits qui font l'objet de cette publication et les informations qu'il contient sont régis par la législation américaine en matière de contrôle des exportations et peuvent être soumis au droit d'autres pays dans le domaine des exportations et importations. Les utilisations finales, ou utilisateurs finaux, pour des armes nucléaires, des missiles, des armes chimiques ou biologiques ou pour le nucléaire maritime, directement ou indirectement, sont strictement interdites. Les exportations ou réexportations vers des pays sous embargo des Etats-Unis, ou vers des entités figurant sur les listes d'exclusion d'exportation américaines, y compris, mais de manière non exclusive, la liste de personnes qui font objet d'un ordre de ne pas participer, d'une façon directe ou indirecte, aux exportations des produits ou des services qui sont régis par la législation américaine en matière de contrôle des exportations et la liste de ressortissants spécifiquement désignés, sont rigoureusement interdites.

LA DOCUMENTATION EST FOURNIE "EN L'ETAT" ET TOUTES AUTRES CONDITIONS, DECLARATIONS ET GARANTIES EXPRESSES OU TACITES SONT FORMELLEMENT EXCLUES, DANS LA MESURE AUTORISEE PAR LA LOI APPLICABLE, Y COMPRIS NOTAMMENT TOUTE GARANTIE IMPLICITE RELATIVE A LA QUALITE MARCHANDE, A L'APTITUDE A UNE UTILISATION PARTICULIERE OU A L'ABSENCE DE CONTREFACON.

Contents

Preface	21
Part I Introduction to Message Queue Administration	31
1 Administrative Tasks and Tools	33
Administrative Tasks	33
Administration in a Development Environment	33
Administration in a Production Environment	34
Administration Tools	36
Built-in Administration Tools	36
JMX-Based Administration	38
2 Quick-Start Tutorial	39
Starting the Administration Console	40
Administration Console Online Help	42
Working With Brokers	43
Starting a Broker	43
Adding a Broker to the Administration Console	43
Connecting to a Broker	45
Viewing Connection Services	46
Working With Physical Destinations	48
Creating a Physical Destination	48
Viewing Physical Destination Properties	49
Purging Messages From a Physical Destination	52
Deleting a Physical Destination	52
Working With Object Stores	53
Adding an Object Store	53

Connecting to an Object Store	56
Working With Administered Objects	56
Adding a Connection Factory	56
Adding a Destination	58
Viewing Administered Object Properties	60
Deleting an Administered Object	61
Running the Sample Application	62
▼ To Run the Sample Application	62
 Part II Administrative Tasks	 65
 3 Starting Brokers and Clients	 67
Preparing System Resources	67
Synchronizing System Clocks	67
Setting the File Descriptor Limit	68
Starting Brokers	68
Starting Brokers Interactively	68
Starting Brokers Automatically	69
Deleting a Broker Instance	74
Starting Clients	74
 4 Configuring a Broker	 77
Broker Services	77
Setting Broker Configuration Properties	78
Modifying Configuration Files	78
Setting Configuration Properties from the Command Line	80
 5 Managing a Broker	 83
Command Utility Preliminaries	84
Using the Command Utility	84
Specifying the User Name and Password	84
Specifying the Broker Name and Port	85
Displaying the Product Version	85
Displaying Help	86

Examples	86
Managing Brokers	87
Shutting Down and Restarting a Broker	87
Quiescing a Broker	88
Pausing and Resuming a Broker	89
Updating Broker Properties	89
Viewing Broker Information	91
6 Configuring and Managing Connection Services	93
Configuring Connection Services	93
Port Mapper	95
Thread Pool Management	96
Managing Connection Services	97
Pausing and Resuming a Connection Service	97
Updating Connection Service Properties	98
Viewing Connection Service Information	99
Managing Connections	101
7 Managing Message Delivery	105
Configuring and Managing Physical Destinations	105
Command Utility Subcommands for Physical Destination Management	106
Creating and Destroying Physical Destinations	107
Pausing and Resuming a Physical Destination	110
Purging a Physical Destination	111
Updating Physical Destination Properties	112
Viewing Physical Destination Information	112
Managing Physical Destination Disk Utilization	116
Using the Dead Message Queue	118
Managing Broker System-Wide Memory	119
Managing Durable Subscriptions	121
Managing Transactions	122
8 Configuring Persistence Services	125
Introduction to Persistence Services	125

File-Based Persistence	126
File-Based Persistence Properties	126
Configuring a File-Based Data Store	127
Securing a File-Based Data Store	128
JDBC-Based Persistence	128
JDBC-Based Persistence Properties	128
Configuring a JDBC-Based Data Store	130
Securing a JDBC-Based Data Store	132
Data Store Formats	132
9 Configuring and Managing Security Services	135
Introduction to Security Services	135
Authentication	137
Authorization	138
Encryption	138
User Authentication	139
Using a Flat-File User Repository	139
Using an LDAP User Repository	145
Using JAAS-Based Authentication	148
User Authorization	153
Access Control File Syntax	154
Application of Authorization Rules	156
Authorization Rules for Connection Services	157
Authorization Rules for Physical Destinations	158
Message Encryption	159
Using Self-Signed Certificates	159
Using Signed Certificates	165
Password Files	168
Security Concerns	168
Password File Contents	169
Connecting Through a Firewall	169
▼ To Enable Broker Connections Through a Firewall	170
10 Configuring and Managing Broker Clusters	171
Configuring Broker Clusters	171

The Cluster Configuration File	172
Cluster Configuration Properties	172
Displaying a Cluster Configuration	176
Managing Broker Clusters	178
Managing Conventional Clusters	178
Managing High-Availability Clusters	183
11 Managing Administered Objects	189
Object Stores	189
LDAP Server Object Stores	189
File-System Object Stores	191
Administered Object Attributes	192
Connection Factory Attributes	192
Destination Attributes	199
Using the Object Manager Utility	199
Adding Administered Objects	200
Deleting Administered Objects	203
Listing Administered Objects	203
Viewing Administered Object Information	204
Modifying Administered Object Attributes	204
Using Command Files	205
12 Monitoring Broker Operations	209
Monitoring Services	209
Introduction to Monitoring Tools	210
Configuring and Using Broker Logging	212
Logger Properties	213
Log Message Format	213
Default Logging Configuration	214
Changing the Logging Configuration	215
Using the Command Utility to Display Metrics Interactively	218
imqcmd metrics	219
Metrics Outputs: imqcmd metrics	220
imqcmd query	222
Using the JMX Administration API	223

Using the Java ES Monitoring Console	223
Using the Message-Based Monitoring API	224
Setting Up Message-Based Monitoring	225
Security and Access Considerations	226
Metrics Outputs: Metrics Messages	227
13 Analyzing and Tuning a Message Service	229
About Performance	229
The Performance Tuning Process	229
Aspects of Performance	230
Benchmarks	230
Baseline Use Patterns	231
Factors Affecting Performance	232
Message Delivery Steps	233
Application Design Factors Affecting Performance	234
Message Service Factors Affecting Performance	238
Adjusting Configuration To Improve Performance	242
System Adjustments	242
Broker Memory Management Adjustments	245
Client Runtime Message Flow Adjustments	246
Adjusting Multiple-Consumer Queue Delivery	248
14 Troubleshooting	251
A Client Cannot Establish a Connection	251
Connection Throughput Is Too Slow	256
A Client Cannot Create a Message Producer	257
Message Production Is Delayed or Slowed	258
Messages Are Backlogged	261
Broker Throughput Is Sporadic	264
Messages Are Not Reaching Consumers	266
Dead Message Queue Contains Messages	267
▼ To Inspect the Dead Message Queue	272

Part III	Reference	275
15	Command Line Reference	277
	Command Line Syntax	277
	Broker Utility	278
	Command Utility	282
	General Command Utility Options	284
	Broker Management	285
	Connection Service Management	288
	Connection Management	288
	Physical Destination Management	289
	Durable Subscription Management	291
	Transaction Management	291
	JMX Management	292
	Object Manager Utility	292
	Database Manager Utility	294
	User Manager Utility	295
	Service Administrator Utility	297
	Key Tool Utility	298
16	Broker Properties Reference	299
	Connection Properties	299
	Routing and Delivery Properties	302
	Persistence Properties	307
	File-Based Persistence Properties	307
	JDBC-Based Persistence Properties	309
	Security Properties	310
	Monitoring Properties	316
	Cluster Configuration Properties	321
	JMX Properties	324
	Alphabetical List of Broker Properties	326
17	Physical Destination Property Reference	333
	Physical Destination Properties	333

18	Administered Object Attribute Reference	337
	Connection Factory Attributes	337
	Connection Handling	337
	Client Identification	341
	Reliability and Flow Control	341
	Queue Browser and Server Sessions	343
	Standard Message Properties	344
	Message Header Overrides	344
	Destination Attributes	345
19	JMS Resource Adapter Property Reference	347
	ResourceAdapter JavaBean	347
	ManagedConnectionFactory JavaBean	350
	ActivationSpec JavaBean	351
20	Metrics Information Reference	355
	JVM Metrics	356
	Brokerwide Metrics	356
	Connection Service Metrics	358
	Physical Destination Metrics	359
21	JES Monitoring Framework Reference	365
	Common Attributes	365
	Message Queue Product Information	366
	Broker Information	366
	Port Mapper Information	367
	Connection Service Information	367
	Destination Information	369
	Persistent Store Information	370
	User Repository Information	370

Part IV	Appendixes	371
A	Platform-Specific Locations of Message Queue Data	373
	Solaris OS	373
	Linux	375
	Windows	376
B	Stability of Message Queue Interfaces	379
	Classification Scheme	379
	Interface Stability	380
C	HTTP/HTTPS Support	383
	HTTP/HTTPS Support Architecture	383
	Enabling HTTP/HTTPS Support	384
	Step 1 (HTTPS Only): Generating a Self-Signed Certificate for the Tunnel Servlet	385
	Step 2 (HTTPS Only): Specifying the Key Store Location and Password	387
	Step 3 (HTTPS Only): Validating and Installing the Server's Self-Signed Certificate	388
	Step 4 (HTTP and HTTPS): Deploying the Tunnel Servlet	392
	Step 5 (HTTP and HTTPS): Configuring the Connection Service	394
	Step 6 (HTTP and HTTPS): Configuring a Connection	395
	Troubleshooting	398
	Server or Broker Failure	398
	Client Failure to Connect Through the Tunnel Servlet	398
D	JMX Support	399
	JMX Connection Infrastructure	399
	MBean Access Mechanism	399
	The JMX Service URL	400
	The Admin Connection Factory	401
	JMX Configuration	402
	RMI Registry Configuration	402
	SSL-Based JMX Connections	406
	JMX Connections Through a Firewall	407

- E Frequently Used Command Utility Commands 409**
 - Syntax 409
 - Broker and Cluster Management 409
 - Broker Configuration Properties (-o option) 410
 - Service and Connection Management 410
 - Durable Subscriber Management 411
 - Transaction Management 411
 - Destination Management 411
 - Destination Configuration Properties (-o option) 411
 - Metrics 412

- Index 413**

Figures

FIGURE 1-1	Local and Remote Administration Utilities	37
FIGURE 2-1	Administration Console Window	41
FIGURE 2-2	Administration Console Help Window	42
FIGURE 2-3	Add Broker Dialog Box	44
FIGURE 2-4	Broker Displayed in Administration Console Window	45
FIGURE 2-5	Connect to Broker Dialog Box	46
FIGURE 2-6	Viewing Connection Services	47
FIGURE 2-7	Service Properties Dialog Box	47
FIGURE 2-8	Add Broker Destination Dialog Box	49
FIGURE 2-9	Broker Destination Properties Dialog Box	51
FIGURE 2-10	Durable Subscriptions Panel	52
FIGURE 2-11	Add Object Store Dialog Box	54
FIGURE 2-12	Object Store Displayed in Administration Console Window	55
FIGURE 2-13	Add Connection Factory Object Dialog Box	57
FIGURE 2-14	Add Destination Object Dialog Box	59
FIGURE 2-15	Destination Object Displayed in Administration Console Window	60
FIGURE 4-1	Broker Configuration Files	79
FIGURE 6-1	Message Queue Connection Services	94
FIGURE 8-1	Persistent Data Stores	126
FIGURE 9-1	Security Support	136
FIGURE 9-2	JAAS Elements	149
FIGURE 9-3	How Message Queue Uses JAAS	150
FIGURE 9-4	Setting Up JAAS Support	151
FIGURE 12-1	Monitoring Services Support	210
FIGURE 13-1	Message Delivery Through a Message Queue Service	233
FIGURE 13-2	Transport Protocol Speeds	240
FIGURE C-1	HTTP/HTTPS Support Architecture	384
FIGURE D-1	Basic JMX Infrastructure	400

FIGURE D-2	Obtaining a Connector Stub from an RMI Registry	401
FIGURE D-3	Obtaining a Connector Stub from an Admin Connection Factory	402

Tables

TABLE 6-1	Message Queue Connection Service Characteristics	94
TABLE 6-2	Connection Service Properties Updated by Command Utility	98
TABLE 7-1	Physical Destination Subcommands for the Command Utility	106
TABLE 7-2	Dead Message Queue Treatment of Physical Destination Properties	118
TABLE 9-1	Initial Entries in Flat-File User Repository	141
TABLE 9-2	User Manager Subcommands	141
TABLE 9-3	General User Manager Options	142
TABLE 9-4	Broker Properties for JAAS Support	153
TABLE 9-5	Authorization Rule Elements	154
TABLE 9-6	Commands That Use Passwords	168
TABLE 9-7	Passwords in a Password File	169
TABLE 9-8	Broker Configuration Properties for Static Port Addresses	169
TABLE 10-1	Broker States	176
TABLE 11-1	LDAP Object Store Attributes	190
TABLE 11-2	File-system Object Store Attributes	191
TABLE 12-1	Benefits and Limitations of Metrics Monitoring Tools	211
TABLE 12-2	Logging Levels	214
TABLE 12-3	imqcmd metrics Subcommand Syntax	219
TABLE 12-4	imqcmd metrics Subcommand Options	219
TABLE 12-5	imqcmd query Subcommand Syntax	222
TABLE 12-6	Metrics Topic Destinations	225
TABLE 13-1	Comparison of High-Reliability and High-Performance Scenarios	234
TABLE 15-1	Broker Utility Options	278
TABLE 15-2	Command Utility Subcommands	282
TABLE 15-3	General Command Utility Options	284
TABLE 15-4	Command Utility Subcommands for Broker Management	286
TABLE 15-5	Command Utility Subcommands for Connection Service Management	288
TABLE 15-6	Command Utility Subcommands for Connection Service Management	289

TABLE 15-7	Command Utility Subcommands for Physical Destination Management	289
TABLE 15-8	Command Utility Subcommands for Durable Subscription Management	291
TABLE 15-9	Command Utility Subcommands for Transaction Management	292
TABLE 15-10	Command Utility Subcommand for JMX Management	292
TABLE 15-11	Object Manager Subcommands	292
TABLE 15-12	Object Manager Options	293
TABLE 15-13	Database Manager Subcommands	294
TABLE 15-14	Database Manager Options	295
TABLE 15-15	User Manager Subcommands	296
TABLE 15-16	General User Manager Options	296
TABLE 15-17	Service Administrator Subcommands	297
TABLE 15-18	Service Administrator Options	297
TABLE 16-1	Broker Connection Properties	300
TABLE 16-2	Broker Routing and Delivery Properties	302
TABLE 16-3	Broker Properties for Auto-Created Destinations	303
TABLE 16-4	Global Broker Persistence Property	307
TABLE 16-5	Broker Properties for File-Based Persistence	308
TABLE 16-6	Broker Properties for JDBC-Based Persistence	309
TABLE 16-7	Broker Security Properties	311
TABLE 16-8	Broker Security Properties for LDAP Authentication	313
TABLE 16-9	Broker Security Properties for JAAS Authentication	316
TABLE 16-10	Broker Monitoring Properties	317
TABLE 16-11	Broker Properties for Cluster Configuration	321
TABLE 16-12	Broker Properties for JMX Support	324
TABLE 16-13	Alphabetical List of Broker Properties	327
TABLE 17-1	Physical Destination Properties	333
TABLE 18-1	Connection Factory Attributes for Connection Handling	338
TABLE 18-2	Message Broker Addressing Schemes	340
TABLE 18-3	Message Broker Address Examples	341
TABLE 18-4	Connection Factory Attributes for Client Identification	341
TABLE 18-5	Connection Factory Attributes for Reliability and Flow Control	342
TABLE 18-6	Connection Factory Attributes for Queue Browser and Server Sessions	343
TABLE 18-7	Connection Factory Attributes for Standard Message Properties	344
TABLE 18-8	Connection Factory Attributes for Message Header Overrides	345
TABLE 18-9	Destination Attributes	345
TABLE 19-1	Resource Adapter Properties	348

TABLE 19-2	Managed Connection Factory Properties	350
TABLE 19-3	ActivationSpec Properties	351
TABLE 20-1	JVM Metrics	356
TABLE 20-2	Brokerwide Metrics	356
TABLE 20-3	Connection Service Metrics	358
TABLE 20-4	Physical Destination Metrics	359
TABLE 21-1	JESMF Common Object Attributes	365
TABLE 21-2	JESMF-Accessible Message Queue Product Attributes	366
TABLE 21-3	JESMF-Accessible Message Queue Broker Attributes	366
TABLE 21-4	JESMF-Accessible Message Queue Port Mapper Attributes	367
TABLE 21-5	JESMF-Accessible Message Queue Connection Service Attributes	368
TABLE 21-6	JESMF-Accessible Message Queue Destination Attributes	369
TABLE 21-7	JESMF-Accessible Message Queue Persistent Store Attributes	370
TABLE 21-8	JESMF-Accessible Message Queue User Repository Attributes	370
TABLE A-1	Message Queue Data Locations on Solaris Platform	373
TABLE A-2	Message Queue Data Locations on Linux Platform	375
TABLE A-3	Message Queue Data Locations on Windows Platform	376
TABLE B-1	Interface Stability Classification Scheme	379
TABLE B-2	Stability of Message Queue Interfaces	380
TABLE C-1	Distinguished Name Information Required for a Self-Signed Certificate	386
TABLE C-2	Broker Configuration Properties for the <code>httpjms</code> and <code>httpsjms</code> Connection Services	394
TABLE D-1	Advantages and Disadvantages of Using an RMI Registry	403
TABLE E-1	Broker Configuration Properties (<code>-o</code> option)	410
TABLE E-2	Destination Configuration Properties (<code>-o</code> option)	411

Examples

EXAMPLE 2-1	Output from Sample Application	63
EXAMPLE 3-1	Displaying Broker Service Startup Options	73
EXAMPLE 5-1	Broker Information Listing	91
EXAMPLE 5-2	Broker Metrics Listing	92
EXAMPLE 6-1	Connection Services Listing	99
EXAMPLE 6-2	Connection Service Information Listing	100
EXAMPLE 6-3	Connection Service Metrics Listing	101
EXAMPLE 6-4	Broker Connections Listing	102
EXAMPLE 6-5	Connection Information Listing	102
EXAMPLE 7-1	Wildcard Publisher	109
EXAMPLE 7-2	Wildcard Subscriber	109
EXAMPLE 7-3	Physical Destination Information Listing	114
EXAMPLE 7-4	Physical Destination Metrics Listing	116
EXAMPLE 7-5	Destination Disk Utilization Listing	117
EXAMPLE 7-6	Durable Subscription Information Listing	122
EXAMPLE 7-7	Broker Transactions Listing	123
EXAMPLE 7-8	Transaction Information Listing	123
EXAMPLE 8-1	Broker Properties for MySQL Database	129
EXAMPLE 8-2	Broker Properties for HADB Database	129
EXAMPLE 9-1	Viewing Information for a Single User	145
EXAMPLE 9-2	Viewing Information for All Users	145
EXAMPLE 9-3	Example 1	155
EXAMPLE 9-4	Example 2	155
EXAMPLE 9-5	Example 3	155
EXAMPLE 9-6	Connection Services Listing	165
EXAMPLE 10-1	Configuration Listing for a Conventional Cluster	177
EXAMPLE 10-2	Configuration Listing for a high-availability Cluster	177
EXAMPLE 11-1	Adding a Connection Factory	201

EXAMPLE 11-2	Adding a Destination to an LDAP Object Store	202
EXAMPLE 11-3	Adding a Destination to a File-System Object Store	202
EXAMPLE 11-4	Deleting an Administered Object	203
EXAMPLE 11-5	Listing All Administered Objects	203
EXAMPLE 11-6	Listing Administered Objects of a Specific Type	204
EXAMPLE 11-7	Viewing Administered Object Information	204
EXAMPLE 11-8	Modifying an Administered Object's Attributes	205
EXAMPLE 11-9	Object Manager Command File Syntax	205
EXAMPLE 11-10	Example Command File	206
EXAMPLE 11-11	Partial Command File	206
EXAMPLE 11-12	Using a Partial Command File	207
EXAMPLE C-1	Tunnel Servlet Status Report	397
EXAMPLE D-1	JMX Service URL When Using an RMI Registry	404
EXAMPLE D-2	JMX Service URL When Not Using an RMI Registry	405
EXAMPLE D-3	JMX Configuration for Firewall When Not Using a RMI Registry	407
EXAMPLE D-4	JMX Configuration for Firewall When Using an RMI Registry	407

Preface

This *Sun Java™ System Message Queue 4.2 Administration Guide* provides background and information needed by system administrators to set up and manage a Sun Java System Message Queue messaging system.

This preface consists of the following sections:

- “Who Should Use This Book” on page 21
- “Before You Read This Book” on page 21
- “How This Book Is Organized” on page 22
- “Documentation Conventions” on page 23
- “Related Documentation” on page 26
- “Sun Welcomes Your Comments” on page 30

Who Should Use This Book

This guide is intended for administrators and application developers who need to perform Message Queue administrative tasks. A Message Queue *administrator* is responsible for setting up and managing a Message Queue messaging system, especially the message broker at the heart of the system.

Before You Read This Book

Before reading this guide, you should read the *Sun Java System Message Queue 4.2 Technical Overview* to become familiar with Message Queue’s implementation of the Java Message Service specification, with the components of the Message Queue service, and with the basic process of developing, deploying, and administering a Message Queue application.

How This Book Is Organized

Table P-1 describes the contents of this manual.

TABLE P-1 Contents of This Manual

Chapter/Appendix	Description
Part I	
Chapter 1, “Administrative Tasks and Tools”	Introduces Message Queue administrative tasks and tools.
Chapter 2, “Quick-Start Tutorial”	Provides a hands-on tutorial to acquaint you with the Message Queue Administration Console.
Part II	
Chapter 3, “Starting Brokers and Clients”	Describes how to start the Message Queue broker and clients.
Chapter 4, “Configuring a Broker”	Describes how configuration properties are set and read, and gives an introduction to the configurable aspects of the broker.
Chapter 5, “Managing a Broker”	Describes broker management tasks.
Chapter 6, “Configuring and Managing Connection Services”	Describes configuration and management tasks relating to the broker's connection services.
Chapter 7, “Managing Message Delivery”	Describes how to create and manage physical destinations and manage other aspects of message delivery.
Chapter 8, “Configuring Persistence Services”	Describes how to set up a file-based or JDBC-based data store to perform persistence services.
Chapter 11, “Managing Administered Objects”	Describes the object store and shows how to perform tasks related to administered objects (connection factories and destinations).
Chapter 10, “Configuring and Managing Broker Clusters”	Describes how to set up and manage a cluster of Message Queue brokers.
Chapter 9, “Configuring and Managing Security Services”	Describes security-related tasks, such as managing password files, authentication, authorization, and encryption.
Chapter 12, “Monitoring Broker Operations”	Describes how to set up and use Message Queue monitoring facilities.
Chapter 11, “Analyzing and Tuning a Message Service”	Describes techniques for analyzing and optimizing message service performance.
Chapter 12, “Troubleshooting”	Provides suggestions for determining the cause of common Message Queue problems and the actions you can take to resolve them.
Part III	

TABLE P-1 Contents of This Manual (Continued)

Chapter/Appendix	Description
Chapter 13, “Command Line Reference”	Provides syntax and descriptions for Message Queue command line utilities.
Chapter 14, “Broker Properties Reference”	Describes the configuration properties of Message Queue message brokers.
Chapter 15, “Physical Destination Property Reference”	Describes the configuration properties of physical destinations.
Chapter 16, “Administered Object Attribute Reference”	Describes the configuration properties of administered objects (connection factories and destinations).
Chapter 17, “JMS Resource Adapter Property Reference”	Describes the configuration properties of the Message Queue Resource Adapter for use with an application server.
Chapter 18, “Metrics Reference”	Describes the metric information that a Message Queue message broker can provide for monitoring, tuning, and diagnostic purposes. .
Chapter 19, “JES Monitoring Framework Reference”	Lists Message Queue attributes that are accessible by means of the Java Enterprise System Monitoring Framework (JESMF).
Part IV	
Appendix A, “Platform-Specific Locations of Message Queue Data”	Lists the locations of Message Queue files on each supported platform.
Appendix B, “Stability of Message Queue Interfaces”	Describes the stability of various Message Queue interfaces.
Appendix C, “HTTP/HTTPS Support”	Describes how to set up and use the Hypertext Transfer Protocol (HTTP) for Message Queue communication.
Appendix D, “JMX Support”	Describes Message Queue’s administrative support for client programs using the Java Management Extensions (JMX) application programming interface
Appendix E, “Frequently Used Command Utility Commands”	Lists some frequently used Message Queue Command utility (imqcmd) commands.

Documentation Conventions

This section describes various conventions used in Message Queue documentation.

Typographic Conventions

Table P-2 shows the typographic conventions used in Message Queue documentation.

TABLE P-2 Typographic Conventions

Typeface	Meaning	Examples
AaBbCc123	Names of commands, files, and directories, and onscreen computer output	Edit your <code>.login</code> file. Use <code>ls -a</code> to list all files. <code>machine_name%</code> you have mail.
AaBbCc123	What you type, contrasted with onscreen computer output	<code>machine_name%</code> su Password:
<i>AaBbCc123</i>	Placeholder: replace with a real name or value	The command to remove a file is <i>rm fileName</i> .
<i>AaBbCc123</i>	Book titles, new terms, and emphasized words	Read Chapter 6 in the <i>User's Guide</i> . <i>A cache</i> is a copy that is stored locally. Do <i>not</i> save the file. Note – Some emphasized items appear online in boldface .

Symbol Conventions

Table P-3 shows symbol conventions used in Message Queue documentation.

TABLE P-3 Symbol Conventions

Symbol	Description	Example	Meaning
[]	Encloses optional arguments and command options	<code>ls [-l]</code>	The <code>-l</code> option is optional.
{ }	Encloses a set of choices for a required command option	<code>-d {y n}</code>	The <code>-d</code> option requires that you use either the <code>y</code> argument or the <code>n</code> argument.
<code>\${ }</code>	Indicates a variable reference	<code>\${com.sun.javaRoot}</code>	References the value of the variable <code>com.sun.javaRoot</code> .
-	Joins simultaneous multiple keystrokes	Ctrl-A	Hold down the Control key while pressing the A key.
+	Joins consecutive multiple keystrokes	Ctrl+A+N	Press the Control key, release it, and then press the subsequent keys.

TABLE P-3 Symbol Conventions (Continued)

Symbol	Description	Example	Meaning
→	Indicates hierarchical menu selection in a graphical user interface	File → New → Templates	From the File menu, choose New; from the New submenu, choose Templates.

Directory Variable Conventions

Message Queue makes use of three directory variables; how they are set varies from platform to platform. Table P-4 describes these variables and how they are used on the Solaris™, Linux, and Windows platforms.

Note – In this manual, these directory variables are shown without platform-specific environment variable notation or syntax (such as \$IMQ_HOME on UNIX). Non-platform-specific pathnames use UNIX directory separator (/) notation.

TABLE P-4 Directory Variable Conventions

Variable	Description
IMQ_HOME	<p>Message Queue home directory:</p> <ul style="list-style-type: none"> ■ Unused on Solaris and Linux; there is no Message Queue home directory. ■ On Windows, denotes the directory <i>mqInstallHome</i>\mq, where <i>mqInstallHome</i> is the installation home directory specified when the product was installed (by default, C:\Program Files\Sun\MessageQueue). <p>Note – The information above applies only to the standalone installation of Message Queue. When Message Queue is installed and run as part of a Sun Java System Application Server installation, IMQ_HOME is set to <i>appServerInstallDir</i>/mq, where <i>appServerInstallDir</i> is the Application Server installation directory.</p>

TABLE P-4 Directory Variable Conventions (Continued)

Variable	Description
IMQ_VARHOME	<p>Directory in which Message Queue temporary or dynamically created configuration and data files are stored; can be set as an environment variable to point to any directory.</p> <ul style="list-style-type: none"> ■ On Solaris, defaults to <code>/var/imq</code>. ■ On Linux, defaults to <code>/var/opt/sun/mq</code>. ■ On Windows, defaults to <code>IMQ_HOME\var</code>. <p>Note – The information above applies only to the standalone installation of Message Queue. When Message Queue is installed and run as part of a Sun Java System Application Server installation, <code>IMQ_VARHOME</code> is set to <code>appServerDomainDir/imq</code>, where <code>appServerDomainDir</code> is the domain directory for the domain starting the Message Queue broker.</p>
IMQ_JAVAHOME	Location of the Java runtime environment (JRE) used by Message Queue executables.

Related Documentation

The information resources listed in this section provide further information about Message Queue in addition to that contained in this manual.

Message Queue Documentation Set

The documents that comprise the Message Queue documentation set are listed in the following table in the order in which you might normally use them. These documents are available through the Sun documentation Web site at

<http://www.sun.com/documentation/>

Click “Software,” followed by “Application & Integration Services,” and then “Message Queue.”

TABLE P-5 Message Queue Documentation Set

Document	Audience	Description
<i>Sun Java System Message Queue 4.2 Technical Overview</i>	Developers and administrators	Describes Message Queue concepts, features, and components.
<i>Sun Java System Message Queue 4.2 Release Notes</i>	Developers and administrators	Includes descriptions of new features, limitations, and known bugs, as well as technical notes.

TABLE P-5 Message Queue Documentation Set (Continued)

Document	Audience	Description
<i>Sun Java System Message Queue 4.2 Installation Guide</i>	Developers and administrators	Explains how to install Message Queue software on Solaris, Linux, and Windows platforms.
<i>Sun Java System Message Queue 4.2 Developer's Guide for Java Clients</i>	Developers	Provides a quick-start tutorial and programming information for developers of Java client programs using the Message Queue implementation of the JMS or SOAP/JAXM APIs.
<i>Sun Java System Message Queue 4.2 Administration Guide</i>	Administrators, also recommended for developers	Provides background and information needed to perform administration tasks using Message Queue administration tools.
<i>Sun Java System Message Queue 4.2 Developer's Guide for C Clients</i>	Developers	Provides programming and reference documentation for developers of C client programs using the Message Queue C implementation of the JMS API (C-API).
<i>Sun Java System Message Queue 4.2 Developer's Guide for JMX Clients</i>	Administrators	Provides programming and reference documentation for developers of JMX client programs using the Message Queue JMX API.

Java Message Service (JMS) Specification

The Message Queue message service conforms to the Java Message Service (JMS) application programming interface, described in the *Java Message Service Specification*. This document can be found at the URL

<http://java.sun.com/products/jms/docs.html>

JavaDoc

JMS and Message Queue API documentation in JavaDoc format is included in your Message Queue installation at the locations shown in Table P-6, depending on your platform. This documentation can be viewed in any HTML browser. It includes standard JMS API documentation as well as Message Queue-specific APIs.

TABLE P-6 JavaDoc Locations

Platform	Location
Solaris	/usr/share/javadoc/imq/index.html

TABLE P-6 JavaDoc Locations (Continued)

Platform	Location
Linux	/opt/sun/mq/javadoc/index.html
Windows	IMQ_HOME\javadoc\index.html where IMQ_HOME is the Message Queue home directory

Example Client Applications

Message Queue provides a number of example client applications to assist developers.

Example Java Client Applications

Example Java client applications are located in the following directories, depending on platform. See the README files located in these directories and their subdirectories for descriptive information about the example applications.

Platform	Location
Solaris	/usr/demo/imq/
Linux	/opt/sun/mq/examples
Windows	IMQ_HOME/demo/ where IMQ_HOME is the Message Queue home directory

Example C Client Programs

Example C client applications are located in the following directories, depending on platform. See the README files located in these directories and their subdirectories for descriptive information about the example applications.

Platform	Location
Solaris	/opt/SUNWimq/demo/C/
Linux	/opt/sun/mq/examples/C/
Windows	IMQ_HOME/demo/C/ where IMQ_HOME is the Message Queue home directory

Example JMX Client Programs

Example Java Management Extensions (JMX) client applications are located in the following directories, depending on platform. See the README files located in these directories and their subdirectories for descriptive information about the example applications.

Platform	Location
Solaris	/opt/SUNWimq/demo/imq/jmx
Linux	/opt/sun/mq/examples/jmx
Windows	IMQ_HOME\demo\jmx where IMQ_HOME is the Message Queue home directory

Online Help

Online help is available for the Message Queue command line utilities; see [Chapter 15](#), “[Command Line Reference](#),” for details. The Message Queue graphical user interface (GUI) administration tool, the Administration Console, also includes a context-sensitive help facility; see “[Administration Console Online Help](#)” on [page 42](#).

Documentation, Support, and Training

The Sun Web site provides information about the following additional resources:

- [Documentation](http://www.sun.com/documentation/) (<http://www.sun.com/documentation/>)
- [Support](http://www.sun.com/support/) (<http://www.sun.com/support/>)
- [Training](http://www.sun.com/training/) (<http://www.sun.com/training/>)

Third-Party Web Site References

Where relevant, this manual refers to third-party URLs that provide additional, related information.

Note – Sun is not responsible for the availability of third-party Web sites mentioned in this manual. Sun does not endorse and is not responsible or liable for any content, advertising, products, or other materials available on or through such sites or resources. Sun will not be responsible or liable for any actual or alleged damage or loss caused or alleged to be caused by or in connection with the use of or reliance on any such content, goods, or services available on or through such sites or resources.

Sun Welcomes Your Comments

Sun is always interested in improving its documentation and welcomes your comments and suggestions. To share your comments, go to the Sun documentation Web site at

<http://docs.sun.com>

and click **Send comments**. In the resulting online form, provide the document title and part number along with your comment. (The part number is a 7-digit or 9-digit number that can be found on the book's title page or in the document's URL. For example, the part number of this book is 819-4467.)



PART I

Introduction to Message Queue Administration

- Chapter 1, “Administrative Tasks and Tools”
- Chapter 2, “Quick-Start Tutorial”

Administrative Tasks and Tools

This chapter provides an overview of Sun Java™ System Message Queue administrative tasks and the tools for performing them, focusing on common features of the command line administration utilities. It consists of the following sections:

- [“Administrative Tasks” on page 33](#)
- [“Administration Tools” on page 36](#)

Administrative Tasks

The typical administrative tasks to be performed depend on the nature of the environment in which you are running Message Queue. The demands of a software development environment in which Message Queue applications are being developed and tested are different from those of a production environment in which such applications are deployed to accomplish useful work. The following sections summarize the typical administrative requirements of these two different types of environment.

Administration in a Development Environment

In a development environment, the emphasis is on flexibility. The Message Queue message service is needed principally for testing applications under development. Administration is generally minimal, with programmers often administering their own systems. Such environments are typically distinguished by the following characteristics:

- Simple startup of brokers for use in testing
- Administered objects instantiated in client code rather than created administratively
- Auto-created destinations
- File-system object store
- File-based persistence

- File-based user repository
- No master broker in multiple-broker clusters

Administration in a Production Environment

In a production environment in which applications must be reliably deployed and run, administration is more important. Administrative tasks to be performed depend on the complexity of the messaging system and of the applications it must support. Such tasks can be classified into two general categories: setup operations and maintenance operations.

Setup Operations

Administrative setup operations in a production environment typically include some or all of the following:

Administrator security

- Setting the password for the default administrative user (admin) ([“Changing a User’s Password” on page 143](#))
- Controlling individual or group access to the administrative connection service ([“Authorization Rules for Connection Services” on page 157](#)) and the dead message queue ([“Authorization Rules for Physical Destinations” on page 158](#))
- Regulating administrative group access to a file-based or Lightweight Directory Access Protocol (LDAP) user repository ([“User Groups and Status” on page 139](#), [“Using an LDAP User Repository” on page 145](#))

General security

- Managing the contents of a file-based user repository ([“Using the User Manager Utility” on page 141](#)) or configuring the broker to use an existing LDAP user repository ([“Using an LDAP User Repository” on page 145](#))
- Controlling the operations that individual users or groups are authorized to perform ([“User Authorization” on page 153](#))
- Setting up encryption services using the Secure Socket Layer (SSL) ([“Message Encryption” on page 159](#))

Administered objects

- Setting up and configuring an LDAP object store ([“LDAP Server Object Stores” on page 189](#))
- Creating connection factories and destinations ([“Adding Administered Objects” on page 200](#))

Broker clusters

- Creating a cluster configuration file ([“The Cluster Configuration File” on page 172](#))
- Designating a master broker ([“Managing a Conventional Cluster's Configuration Change Record” on page 182](#))

Persistence

- Configuring a broker to use a persistent store ([Chapter 8, “Configuring Persistence Services”](#)).

Memory management

- Setting a destination's configuration properties to optimize its memory usage ([“Updating Physical Destination Properties” on page 112](#), [Chapter 17, “Physical Destination Property Reference”](#))

Maintenance Operations

Because application performance, reliability, and security are at a premium in production environments, message service resources must be tightly monitored and controlled through ongoing administrative maintenance operations, including the following:

Broker administration and tuning

- Using broker metrics to tune and reconfigure a broker ([Chapter 13, “Analyzing and Tuning a Message Service”](#))
- Managing broker memory resources ([“Managing Broker System-Wide Memory” on page 119](#))
- Creating and managing broker clusters to balance message load ([Chapter 10, “Configuring and Managing Broker Clusters”](#))
- Recovering failed brokers ([“Starting Brokers” on page 68](#)).

Administered objects

- Adjusting connection factory attributes to ensure the correct behavior of client applications ([“Connection Factory Attributes” on page 192](#))
- Monitoring and managing physical destinations ([“Configuring and Managing Physical Destinations” on page 105](#))
- Controlling user access to destinations ([“Authorization Rules for Physical Destinations” on page 158](#))

Client management

- Monitoring and managing durable subscriptions (see [“Managing Durable Subscriptions” on page 121](#)).
- Monitoring and managing transactions (see [“Managing Transactions” on page 122](#)).

Administration Tools

This section describes the tools you use to configure and manage Message Queue broker services. The tools fall into two categories:

- [“Built-in Administration Tools” on page 36](#)
- [“JMX-Based Administration” on page 38](#)

Built-in Administration Tools

Message Queue's built-in administration tools include both command line and GUI tools:

- [“Command Line Utilities” on page 36](#)
- [“Administration Console” on page 37](#)

Command Line Utilities

All Message Queue utilities are accessible via a command line interface. Utility commands share common formats, syntax conventions, and options. These utilities allow you to perform various administrative tasks, as noted below, and therefore can require different administrative permissions:

- The **Broker utility** (`imqbrokerd`) starts up brokers and specifies their configuration properties, including connecting them together into a cluster. Permissions: User account that initially started the broker.
- The **Command utility** (`imqcmd`) controls brokers and their resources and manages physical destinations. Permissions: Message Queue admin user account.
- The **Object Manager utility** (`imqobjmgr`) manages provider-independent *administered objects* in an object store accessible via the Java Naming and Directory Interface (JNDI). Permissions: Object store (flat-file or LDAP server) access account.
- The **Database Manager utility** (`imqdbmgr`) creates and manages databases for persistent storage that conform to the Java Database Connectivity (JDBC) standard. Permissions: JDBC database manager account.
- The **User Manager utility** (`imqusermgr`) populates a file-based user repository for user authentication and authorization. Permissions: user account that initially started the broker.
- The **Service Administrator utility** (`imqsvcadm`) installs and manages a broker as a Windows service. Permissions: Administrator.
- The **Key Tool utility** (`imqkeytool`) generates self-signed certificates for Secure Socket Layer (SSL) authentication. Permissions: root user (Solaris and Linux platforms) or Administrator (Windows platform).

The executable files for the above command line utilities are in the `/bin` directory shown in [Appendix A, “Platform-Specific Locations of Message Queue Data.”](#)

See [Chapter 15, “Command Line Reference,”](#) for detailed information on the use of these utilities.

Administration Console

The Message Queue *Administration Console* combines some of the capabilities of the Command and Object Manager utilities. You can use it to perform the following tasks:

- Connect to and control a broker remotely
- Create and manage physical destinations
- Create and manage administered objects in a JNDI object store

However, you cannot use the Administration Console to perform such tasks as starting up a broker, creating broker clusters, managing a JDBC database or a user repository, installing a broker as a Windows service, or generating SSL certificates. For these, you need the other command line utilities (Broker, Database Manager, User Manager, Service Administrator, and Key Tool), which cannot operate remotely and must be run on the same host as the broker they manage (see [Figure 1-1](#)).

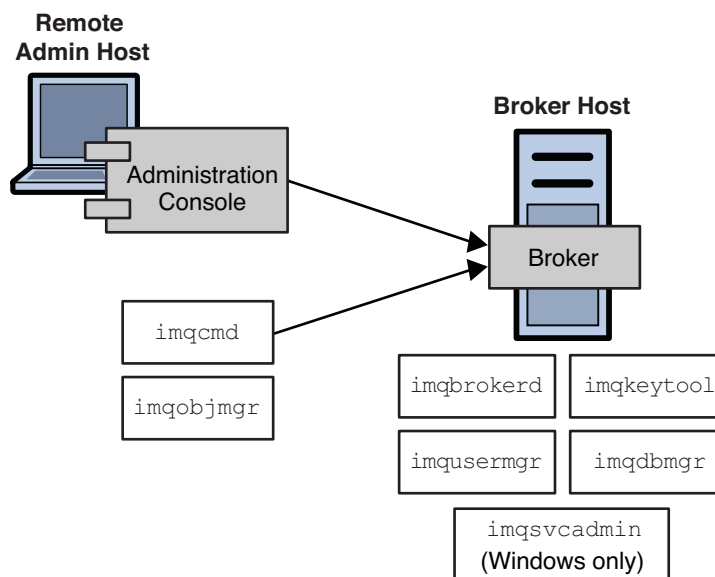


FIGURE 1-1 Local and Remote Administration Utilities

See [Chapter 2, “Quick-Start Tutorial,”](#) for a brief, hands-on introduction to the Administration Console. More detailed information on its use is available through its own help facility.

JMX-Based Administration

To serve customers who need a standard programmatic means to monitor and access the broker, Message Queue also supports the Java Management Extensions (JMX) architecture, which allows a Java application to manage broker resources programmatically.

- *Resources* include everything that you can manipulate using the Command utility (`imqcmd`) and the Message Queue Admin Console: the broker, connection services, connections, destinations, durable subscribers, transactions, and so on.
- *Management* includes the ability to dynamically configure and monitor resources, and the ability to obtain notifications about state changes and error conditions.

JMX is the Java standard for building management applications. Message Queue is based on the JMX 1.2 specification, which is part of JDK 1.5.

For information on the broker's JMX infrastructure and how to configure the broker to support JMX client applications,, see [Appendix D, “JMX Support.”](#)

To manage a Message Queue broker using the JMX architecture, see [Sun Java System Message Queue 4.2 Developer's Guide for JMX Clients](#).

Quick-Start Tutorial

This quick-start tutorial provides a brief introduction to Message Queue administration by guiding you through some basic administrative tasks using the Message Queue Administration Console, a graphical interface for administering a message broker and object store. The chapter consists of the following sections:

- “Starting the Administration Console” on page 40
- “Administration Console Online Help” on page 42
- “Working With Brokers” on page 43
- “Working With Physical Destinations” on page 48
- “Working With Object Stores” on page 53
- “Working With Administered Objects” on page 56
- “Running the Sample Application” on page 62

The tutorial sets up the physical destinations and administered objects needed to run a simple JMS-compliant application, `HelloWorldMessageJNDI`. The application is available in the `helloworld` subdirectory of the example applications directory (demo on the Solaris and Windows platforms or examples on Linux; see [Appendix A, “Platform-Specific Locations of Message Queue Data”](#)). In the last part of the tutorial, you will run this application.

Note – You must have the Message Queue product installed in order to follow the tutorial. If necessary, see the *Message Queue Installation Guide* for instructions.

The tutorial is only a basic introduction; it is not a substitute for reading the documentation. By following the steps described in the tutorial, you will learn how to

- Start a Message Queue broker
- Connect to a broker and use the Administration Console to manage it
- Create physical destinations on the broker
- Create an object store and use the Administration Console to connect to it
- Add administered objects to the object store and view their properties

Note – The instructions given in this tutorial are specific to the Windows platform. Where necessary, supplemental notes are added for users of other platforms.

Some administrative tasks cannot be accomplished using the Administration Console. You must use command line utilities to perform such tasks as the following:

- Start up a broker
- Create a broker cluster
- Configure certain physical destination properties
- Manage a JDBC database for persistent storage
- Manage a user repository
- Install a broker as a Windows service
- Generate SSL certificates

All of these tasks are covered in later chapters of this manual.

Starting the Administration Console

To start the Administration Console, use one of the following methods:

- On Solaris, enter the command

```
/usr/bin/imqadmin
```

- On Linux, enter the command

```
/opt/sun/mq/bin/imqadmin
```

- On Windows, navigate to `IMQ_HOME\bin` and run the `imqadmin` executable.

For example, use the Windows Explorer to navigate to `IMQ_HOME\bin` and then double-click `imqadmin`.

You may need to wait a few seconds before the Administration Console window is displayed (see [Figure 2-1](#)).

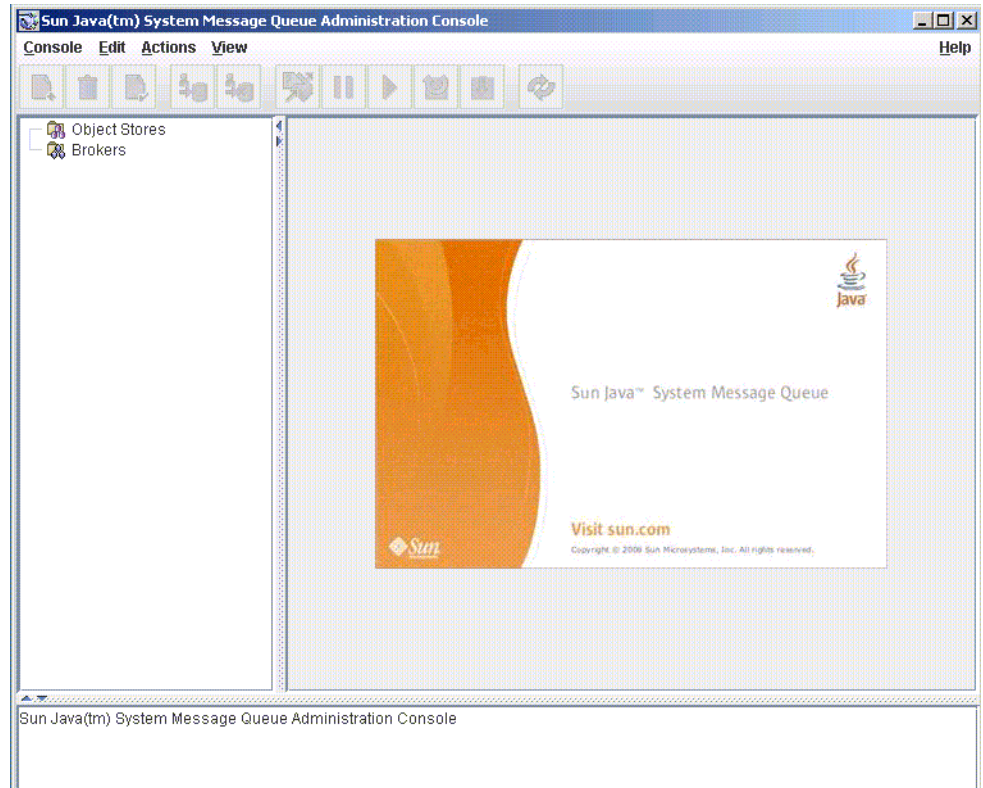


FIGURE 2-1 Administration Console Window

Take a few seconds to examine the Administration Console window. It has a menu bar at the top, a tool bar just below it, a navigation pane to the left, a result pane to the right (now displaying graphics identifying the Sun Java™ System Message Queue product), and a status pane at the bottom.

Note – As you work with the Administration Console, you can use the Refresh command on the View menu to update the visual display of any element or group of elements, such as a list of brokers or object stores.

Administration Console Online Help

The Administration Console provides a help facility containing complete information about how to use the Console to perform administrative tasks. To use the help facility, pull down the Help menu at the right end of the menu bar and choose Overview. The Administration Console's Help window (Figure 2–2) will be displayed.

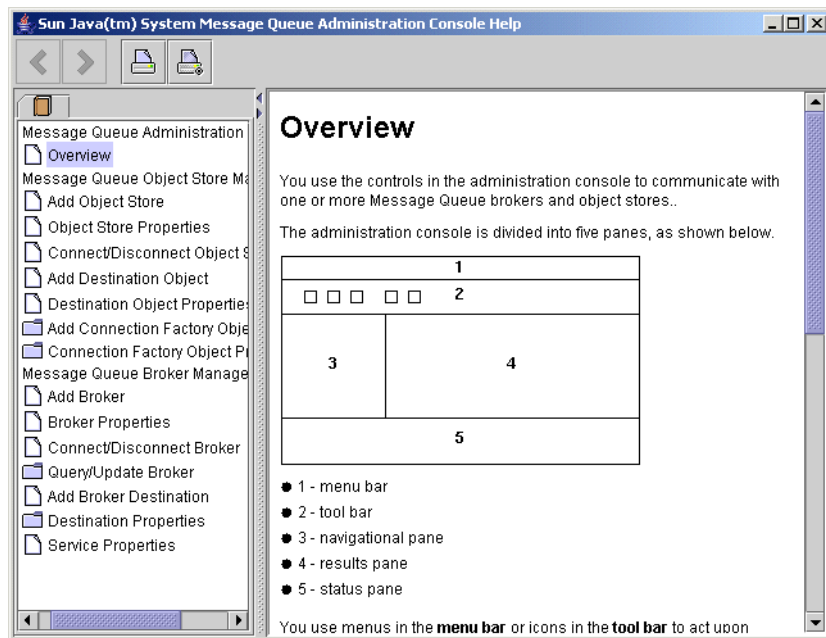


FIGURE 2–2 Administration Console Help Window

The Help window's navigation pane, on the left, organizes topics into three areas: Message Queue Administration Console, Message Queue Object Store Management, and Message Queue Broker Management. Within each area are files and folders. The folders provide help for dialog boxes containing multiple tabs, the files for simple dialog boxes or individual tabs. When you select an item in the navigation pane, the result pane to the right shows the contents of that item. With the Overview item chosen, the result pane displays a skeletal view of the Administration Console window identifying each of the window's panes, as shown in the figure.

Your first task with the Administration Console will be to create a reference to a broker. Before you start, however, check the Help window for information. Click the Add Broker item in the Help window's navigation pane; the contents of the result pane will change to show text explaining what it means to add a broker and describing the use of each field in the Add Broker dialog box. Read through the help text, then close the Help window.

Working With Brokers

This section describes how to use the Administration Console to connect to and manage message brokers.

Starting a Broker

You cannot start a broker using the Administration Console. Instead, use one of the following methods:

- On Solaris, enter the command

```
/usr/bin/imqbrokerd
```

- On Linux, enter the command

```
/opt/sun/mq/bin/imqbrokerd
```

- On Windows, navigate to `IMQ_HOME\bin` and run the `imqbrokerd` executable.

For example, use the Windows Explorer to navigate to `IMQ_HOME\bin` and then double-click `imqbrokerd`.

If you used the Windows Start menu, the command window will appear, indicating that the broker is ready by displaying lines like the following:

```
Loading persistent data...
Broker "imqbroker@stan:7676 ready.
```

Reactivate the Administration Console window. You are now ready to add the broker to the Console and connect to it. You do not have to start the broker before adding a reference to it in the Administration Console, but you must start it before you can connect to it.

Adding a Broker to the Administration Console

Adding a broker creates a reference to that broker in the Administration Console. After adding the broker, you can connect to it.

▼ To Add a Broker to the Administration Console

- 1 Click on the **Brokers** item in the Administration Console window's navigation pane and choose **Add Broker** from the Actions menu.

Alternatively, you can right-click on **Brokers** and choose **Add Broker** from the pop-up context menu. In either case, the Add Broker dialog box ([Figure 2–3](#)) will appear.

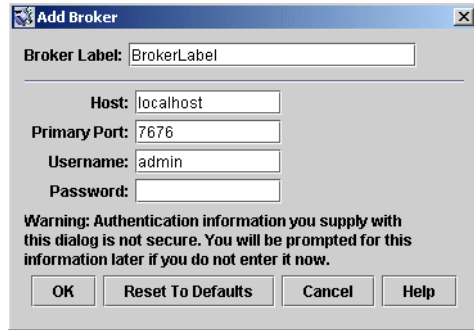


FIGURE 2-3 Add Broker Dialog Box

2 Enter a name for the broker in the Broker Label field.

This provides a label that identifies the broker in the Administration Console.

Note the default host name (`localhost`) and primary port (`7676`) specified in the dialog box. These are the values you must specify later, when you configure the connection factory that the client will use to create connections to this broker.

For this exercise, type the name `MyBroker` into the Broker Label field. Leave the Password field blank; your password will be more secure if you specify it at connection time.

3 Click OK to add the broker and dismiss the dialog box.

The new broker will appear under Brokers in the navigation pane, as shown in [Figure 2-4](#). The red X over the broker's icon indicates that it is not currently connected to the Administration Console.

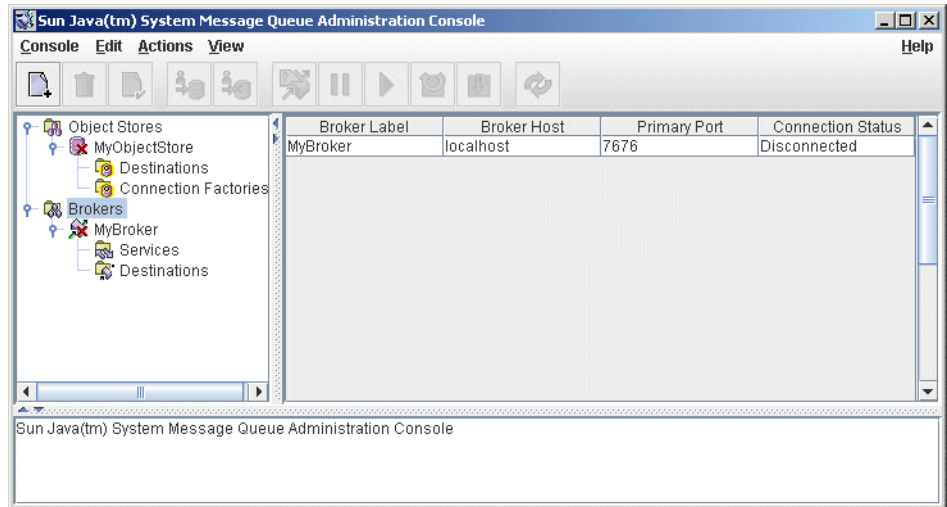


FIGURE 2-4 Broker Displayed in Administration Console Window

Once you have added a broker, you can use the Properties command on the Actions menu (or the pop-up context menu) to display a Broker Properties dialog box, similar to the Add Broker dialog shown in “[Adding a Broker to the Administration Console](#)” on page 43, to view or modify any of its properties.

Connecting to a Broker

Now that you have added a broker to the Administration Console, you can proceed to connect to it.

▼ To Connect to a Broker

- 1 Click on the broker’s name in the Administration Console window’s navigation pane and choose **Connect to Broker** from the Actions menu.

Alternatively, you can right-click on the broker’s name and choose **Connect to Broker** from the pop-up context menu. In either case, the Connect to Broker dialog box ([Figure 2-5](#)) will appear.

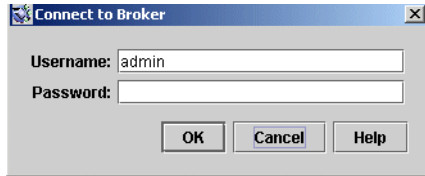


FIGURE 2-5 Connect to Broker Dialog Box

2 Enter the user name and password with which to connect to the broker.

The dialog box initially displays the default user name, `admin`. In a real-world environment, you should establish secure user names and passwords as soon as possible (see [“User Authentication” on page 139](#)); for this exercise, simply use the default value.

The password associated with the default user name is also `admin`; type it into the Password field in the dialog box. This will connect you to the broker with administrative privileges.

3 Click OK to connect to the broker and dismiss the dialog box.

Once you have connected to the broker, you can use the commands on the Actions menu (or the context menu) to perform the following operations on a selected broker:

- Pause Broker temporarily suspends the operation of a running broker.
- Resume Broker resumes the operation of a paused broker.
- Restart Broker reinitializes and restarts a broker.
- Shut Down Broker terminates the operation of a broker.
- Query/Update Broker displays or modifies a broker’s configuration properties.
- Disconnect from Broker terminates the connection between a broker and the Administration Console.

Viewing Connection Services

A broker is distinguished by the connection services it provides and the physical destinations it supports.

▼ To View Available Connection Services

1 Select Services under the broker’s name in the Administration Console window’s navigation pane.

A list of the available services will appear in the result pane (see [Figure 2-6](#)), showing the name, port number, and current state of each service.

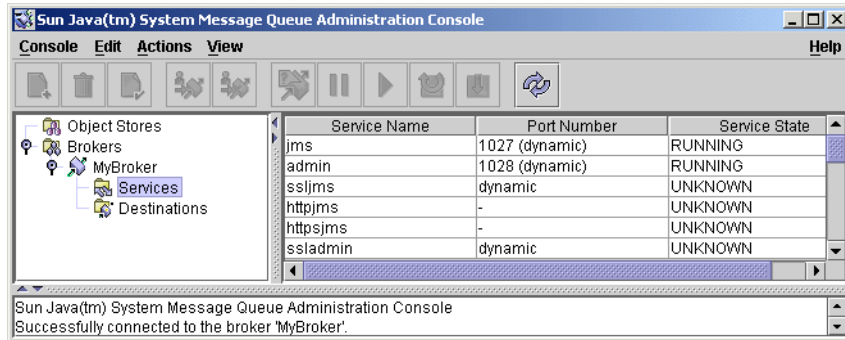


FIGURE 2-6 Viewing Connection Services

- 2 **Select a service by clicking on its name in the result pane.**

For this exercise, select the name `jms`.

- 3 **Choose Properties from the Actions menu.**

The Service Properties dialog box (Figure 2-7) will appear. You can use this dialog box to assign the service a static port number and to change the minimum and maximum number of threads allocated for it.

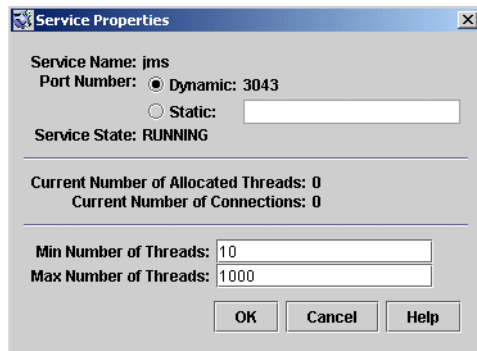


FIGURE 2-7 Service Properties Dialog Box

For this exercise, do not change any of the connection service's properties.

- 4 **Click OK to accept the new property values and dismiss the dialog box.**

The Actions menu also contains commands for pausing and resuming a service. If you select the admin service and pull down the Actions menu, however, you will see that the Pause Service command is disabled. This is because the admin service is the Administration Console's link to the broker: if you paused it, you would no longer be able to access the broker.

Working With Physical Destinations

A *physical destination* is a location on a message broker where messages received from a message producer are held for later delivery to one or more message consumers. Destinations are of two kinds, depending on the *messaging domain* in use: *queues* (point-to-point domain) and *topics* (publish/subscribe domain). See the *Message Queue Technical Overview* for further discussion of messaging domains and the destinations associated with them.

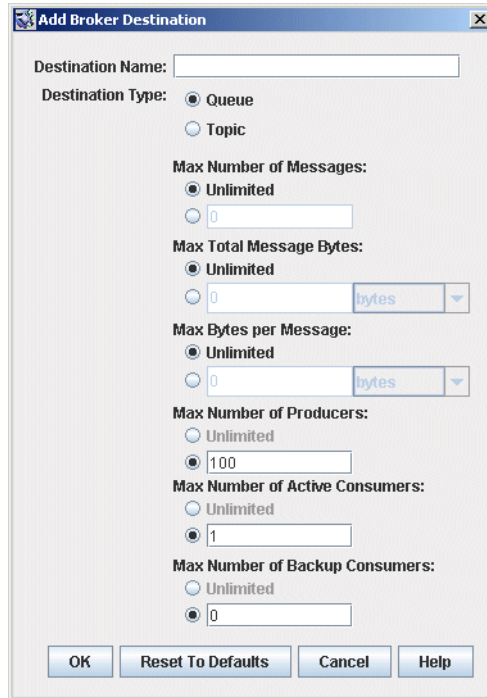
Creating a Physical Destination

By default, message brokers are configured to create new physical destinations automatically whenever a message producer or consumer attempts to access a nonexistent destination. Such *auto-created destinations* are convenient to use while testing client code in a software development environment. In a production setting, however, it is advisable to disable the automatic creation of destinations and instead require all destinations to be created explicitly by an administrator. The following procedure shows how to add such an *admin-created destination* to a broker.

▼ To Add a Physical Destination to a Broker

- 1 Click on the Destinations item under the broker's name in the Administration Console window's navigation pane and choose Add Broker Destination from the Actions menu.

Alternatively, you can right-click on Destinations and choose Add Broker Destination from the pop-up context menu. In either case, the Add Broker Destination dialog box ([Figure 2–8](#)) will appear.



The dialog box is titled "Add Broker Destination". It contains the following fields and controls:

- Destination Name:** A text input field.
- Destination Type:** Two radio buttons: ☒ Queue and ☐ Topic.
- Max Number of Messages:** Two radio buttons: ☒ Unlimited and ☐ 0 (with a text input field).
- Max Total Message Bytes:** Two radio buttons: ☒ Unlimited and ☐ 0 (with a text input field and a "bytes" dropdown menu).
- Max Bytes per Message:** Two radio buttons: ☒ Unlimited and ☐ 0 (with a text input field and a "bytes" dropdown menu).
- Max Number of Producers:** Two radio buttons: ☐ Unlimited and ☒ 100 (with a text input field).
- Max Number of Active Consumers:** Two radio buttons: ☐ Unlimited and ☒ 1 (with a text input field).
- Max Number of Backup Consumers:** Two radio buttons: ☐ Unlimited and ☒ 0 (with a text input field).

At the bottom are four buttons: OK, Reset To Defaults, Cancel, and Help.

FIGURE 2-8 Add Broker Destination Dialog Box

2 Enter a name for the physical destination in the Destination Name field.

Note the name that you assign to the destination; you will need it later when you create an administered object corresponding to this physical destination.

For this exercise, type in the name `MyQueueDest`.

3 Select the Queue or Topic radio button to specify the type of destination to create.

For this exercise, select Queue if it is not already selected.

4 Click OK to add the physical destination and dismiss the dialog box.

The new destination will appear in the result pane.

Viewing Physical Destination Properties

You can use the Properties command on the Administration Console's Actions menu to view or modify the properties of a physical destination.

▼ To View or Modify the Properties of a Physical Destination

- 1 **Select Destinations under the broker's name in the Administration Console window's navigation pane.**

A list of the available physical destinations will appear in the result pane, showing the name, type, and current state of each destination.

- 2 **Select a physical destination by clicking on its name in the result pane.**

- 3 **Choose Properties from the Actions menu.**

The Broker Destination Properties dialog box ([Figure 2–9](#)) will appear, showing current status and configuration information about the selected physical destination. You can use this dialog box to change various configuration properties, such as the maximum number of messages, producers, and consumers that the destination can accommodate.

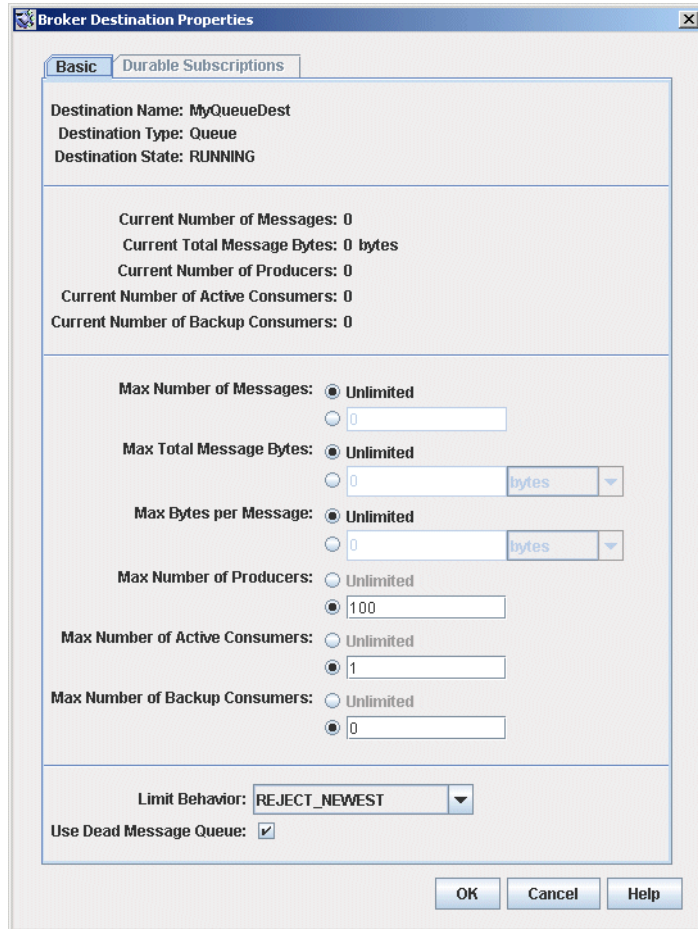


FIGURE 2-9 Broker Destination Properties Dialog Box

For this exercise, do not change any of the destination's properties.

For topic destinations, the Broker Destination Properties dialog box contains an additional tab, Durable Subscriptions. Clicking on this tab displays the Durable Subscriptions panel (Figure 2-10), listing information about all durable subscriptions currently associated with the given topic.

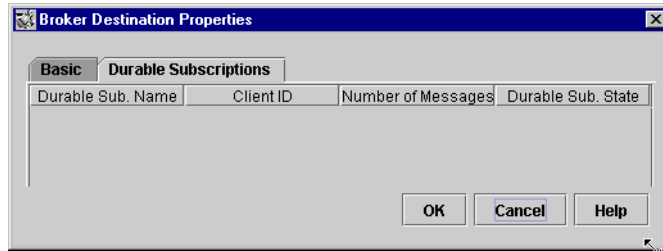


FIGURE 2-10 Durable Subscriptions Panel

You can use the Durable Subscriptions panel's Purge and Delete buttons to

- Purge all pending messages associated with a durable subscription
- Remove a durable subscription from the topic

The Durable Subscriptions tab is disabled for queue destinations.

- 4 Click OK to accept the new property values and dismiss the dialog box.

Purging Messages From a Physical Destination

Purging messages from a physical destination removes all pending messages associated with the destination, leaving the destination empty.

▼ To Purge Messages From a Physical Destination

- 1 **Select Destinations under the broker's name in the Administration Console window's navigation pane.**

A list of the available physical destinations will appear in the result pane, showing the name, type, and current state of each destination.

- 2 **Select a destination by clicking on its name in the result pane.**

- 3 **Choose Purge Messages from the Actions menu.**

A confirmation dialog box will appear, asking you to confirm that you wish to proceed with the operation.

- 4 **Click Yes to confirm the operation and dismiss the confirmation dialog.**

Deleting a Physical Destination

Deleting a destination purges all of its messages and then destroys the destination itself, removing it permanently from the broker to which it belongs.

▼ To Delete a Physical Destination

- 1 **Select Destinations under the broker's name in the Administration Console window's navigation pane.**

A list of the available destinations will appear in the result pane, showing the name, type, and current state of each destination.

- 2 **Select a destination by clicking on its name in the result pane.**

- 3 **Choose Delete from the Edit menu.**

A confirmation dialog box will appear, asking you to confirm that you wish to proceed with the operation.

- 4 **Click Yes to confirm the operation and dismiss the confirmation dialog.**

For this exercise, do not delete the destination `MyQueueDest` that you created earlier; instead, click No to dismiss the confirmation dialog without performing the delete operation.

Working With Object Stores

An *object store* is used to store Message Queue *administered objects*, which encapsulate implementation and configuration information specific to a particular Message Queue provider. An object store can be either a Lightweight Directory Access Protocol (LDAP) directory server or a directory in the local file system.

Although it is possible to instantiate and configure administered objects directly from within a client application's code, it is generally preferable to have an administrator create and configure these objects and store them in an object store, where client applications can access them using the Java Naming and Directory Interface (JNDI). This allows the client code itself to remain provider-independent.

Adding an Object Store

Although the Administration Console allows you to *manage* an object store, you cannot use it to *create* one; the LDAP server or file-system directory that will serve as the object store must already exist ahead of time. You can then add this existing object store to the Administration Console, creating a reference to it that you can use to operate on it from within the Console.

Note – The sample application used in this chapter assumes that the object store is held in a directory named Temp on the C drive. If you do not already have a folder named Temp on your C drive, create one before proceeding with the following exercise. (On non-Windows platforms, you can use the /tmp directory, which should already exist.)

▼ To Add an Object Store to the Administration Console

- 1 Click on the Object Stores item in the Administration Console window's navigation pane and choose Add Object Store from the Actions menu.

Alternatively, you can right-click on Object Stores and choose Add Object Store from the pop-up context menu. In either case, the Add Object Store dialog box (Figure 2–11) will appear.

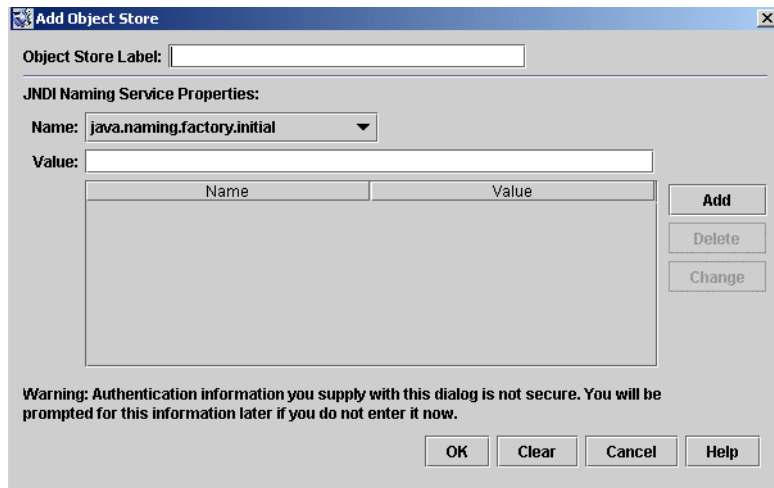


FIGURE 2–11 Add Object Store Dialog Box

- 2 Enter a name for the object store in the Object Store Label field.
This provides a label that identifies the object store in the Administration Console.
For this exercise, type in the name MyObjectStore.
- 3 Enter the JNDI attribute values to be used for looking up administered objects:
 - a. Select the name of the attribute you wish to specify from the Name pull-down menu.
 - b. Type the value of the attribute into the Value field.

c. Click the Add button to add the specified attribute value.

The property and its value will appear in the property summary pane.

Repeat steps “Adding an Object Store” on page 53 to “Adding an Object Store” on page 53 for as many attributes as you need to set.

For this exercise, set the `java.naming.factory.initial` attribute to

```
com.sun.jndi.fscontext.RefFSContextFactory
```

and the `java.naming.provider.url` attribute to

```
file:///C:/Temp
```

(or `file:///tmp` on the Solaris or Linux platforms). These are the only attributes you need to set for a file-system object store; see “LDAP Server Object Stores” on page 189 for information on the attribute values needed for an LDAP store.

4 Click OK to add the object store and dismiss the dialog box.

The new object store will appear under Object Stores in the navigation pane, as shown in Figure 2–12. The red X over the object store’s icon indicates that it is not currently connected to the Administration Console.

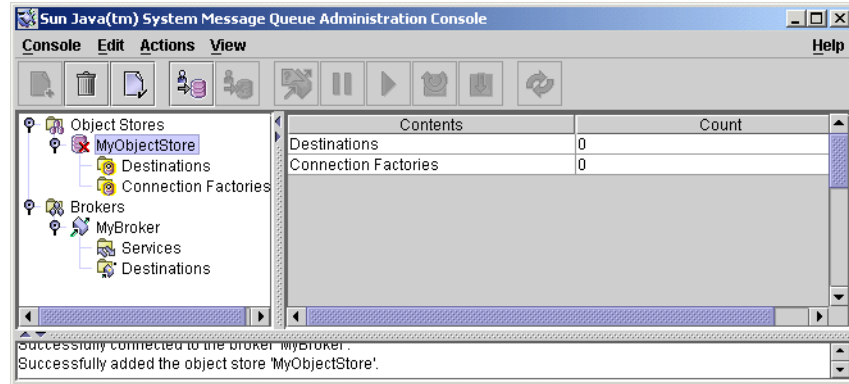


FIGURE 2–12 Object Store Displayed in Administration Console Window

When you click on the object store in the navigation pane, its contents are listed in the result pane. Since you have not yet added any administered objects to the object store, the Count column shows 0 for both destinations and connection factories.

Once you have added an object store, you can use the Properties command on the Actions menu (or the pop-up context menu) to display an Object Store Properties dialog box, similar to the Add Object Store dialog shown in Figure 2–11, to view or modify any of its properties.

Connecting to an Object Store

Now that you have added an object store to the Administration Console, you must connect to it in order to add administered objects to it.

▼ To Connect to an Object Store

- Click on the object store's name in the Administration Console window's navigation pane and choose **Connect to Object Store** from the Actions menu.

Alternatively, you can right-click on the object store's name and choose **Connect to Object Store** from the pop-up context menu. In either case, the red X will disappear from the object store's icon, indicating that it is now connected to the Administration Console.

Working With Administered Objects

Once you have connected an object store to the Administration Console, you can proceed to add administered objects (connection factories and destinations) to it. This section describes how.

Note – The Administration Console displays only Message Queue administered objects. If an object store contains a non-Message Queue object with the same lookup name as an administered object that you want to add, you will receive an error when you attempt the add operation.

Adding a Connection Factory

Connection factories are used by client applications to create connections to a broker. By configuring a connection factory, you can control the properties of the connections it creates.

▼ To Add a Connection Factory to an Object Store

- 1 Make sure the object store is connected to the Administration Console (see [“Connecting to an Object Store” on page 56](#)).
- 2 Click on the **Connection Factories** item under the object store's name in the Administration Console window's navigation pane and choose **Add Connection Factory Object** from the Actions menu.

Alternatively, you can right-click on **Connection Factories** and choose **Add Connection Factory Object** from the pop-up context menu. In either case, the **Add Connection Factory Object** dialog box ([Figure 2-13](#)) will appear.

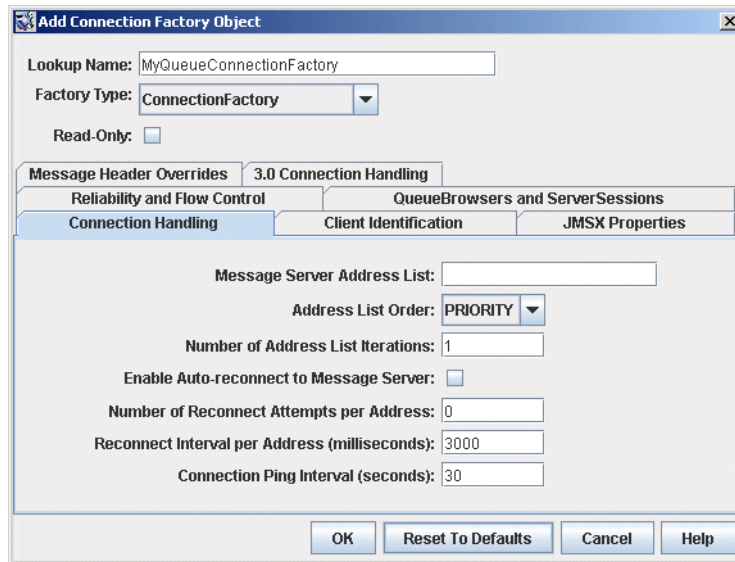


FIGURE 2-13 Add Connection Factory Object Dialog Box

3 Enter a name for the connection factory in the Lookup Name field.

This is the name that client applications will use when looking up the connection factory with JNDI.

For this exercise, type in the name `MyQueueConnectionFactory`.

4 Choose the type of connection factory you wish to create from the Factory Type pull-down menu.

For this exercise, choose `QueueConnectionFactory`.

5 Click the Connection Handling tab.

The Connection Handling panel will appear, as shown in [Figure 2-13](#).

6 Fill in the Message Server Address List field with the address(es) of the broker(s) to which this connection factory will create connections.

The address list may consist of a single broker or (in the case of a broker cluster) multiple brokers. For each broker, it specifies information such as the broker's connection service, host name, and port number. The exact nature and syntax of the information to be specified varies, depending on the connection service to be used; see [“Connection Handling” on page 337](#) for specifics.

For this exercise, there is no need to type anything into the Message Server Address List field, since the sample application `HelloWorldMessageJNDI` expects the connection factory to use the standard address list attributes to which it is automatically configured by default (connection service `jms`, host name `localhost`, and port number `7676`).

7 Configure any other attributes of the connection factory as needed.

The Add Connection Factory Object dialog box contains a number of other panels besides Connection Handling, which can be used to configure various attributes for a connection factory.

For this exercise, do not change any of the other attribute settings. You may find it instructive, however, to click through the other tabs to get an idea of the kinds of configuration information that can be specified. Use the Help button to learn more about the contents of these other configuration panels.

8 If appropriate, click the Read-Only checkbox.

This locks the connection factory object's configuration attributes to the values they were given at creation time. A read-only administered object's attributes cannot be overridden, whether programmatically from client code or administratively from the command line.

For this exercise, do not check Read-Only.

9 Click OK to create the connection factory, add it to the object store, and dismiss the dialog box.

The new connection factory will appear in the result pane.

Adding a Destination

A *destination* administered object represents a physical destination on a broker, enabling clients to send messages to that physical destination independently of provider-specific configurations and naming syntax. When a client sends a message addressed via the administered object, the broker will deliver the message to the corresponding physical destination, if it exists. If no such physical destination exists, the broker will create one automatically if auto-creation is enabled, as described under [“Creating a Physical Destination” on page 48](#), and deliver the message to it; otherwise, it will generate an error signaling that the message cannot be delivered.

The following procedure describes how to add a destination administered object to the object store corresponding to an existing physical destination.

▼ To Add a Destination to an Object Store

- 1 Make sure the object store is connected to the Administration Console (see [“Connecting to an Object Store” on page 56](#)).**

- 2 Click on the Destinations item under the object store's name in the Administration Console window's navigation pane and choose Add Destination Object from the Actions menu.

Alternatively, you can right-click on Destinations and choose Add Destination Object from the pop-up context menu. In either case, the Add Destination Object dialog box (Figure 2–14) will appear.

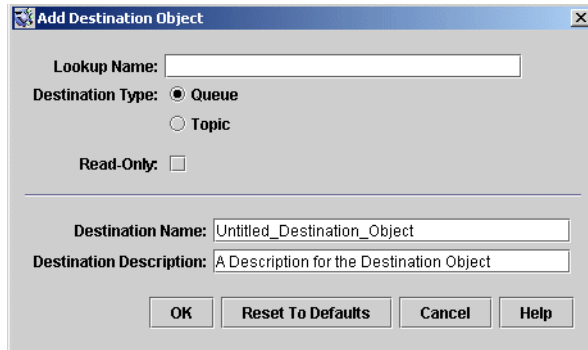


FIGURE 2–14 Add Destination Object Dialog Box

- 3 Enter a name for the destination administered object in the Lookup Name field.

This is the name that client applications will use when looking up the destination with JNDI.

For this exercise, type in the name MyQueue.

- 4 Select the Queue or Topic radio button to specify the type of destination object to create.

For this exercise, select Queue if it is not already selected.

- 5 Enter the name of the corresponding physical destination in the Destination Name field.

This is the name you specified when you added the physical destination to the broker (see “Working With Physical Destinations” on page 48).

For this exercise, type in the name MyQueueDest.

- 6 Optionally, enter a brief description of the destination in the Destination Description field.

The contents of this field are intended strictly for human consumption and have no effect on client operations.

For this exercise, you can either delete the contents of the Destination Description field or type in some descriptive text such as

Example destination for MQ Admin Guide tutorial

7 If appropriate, click the Read-Only checkbox.

This locks the destination object's configuration attributes to the values they were given at creation time. A read-only administered object's attributes cannot be overridden, whether programmatically from client code or administratively from the command line.

For this exercise, do not check Read-Only.

8 Click OK to create the destination object, add it to the object store, and dismiss the dialog box.

The new destination object will appear in the result pane, as shown in [Figure 2–15](#).

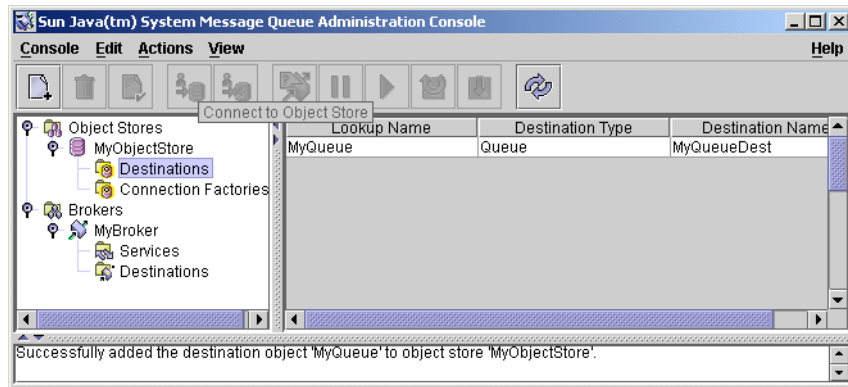


FIGURE 2–15 Destination Object Displayed in Administration Console Window

Viewing Administered Object Properties

You can use the Properties command on the Administration Console's Actions menu to view or modify the properties of an administered object.

▼ To View or Modify the Properties of an Administered Object

1 Select Connection Factories or Destinations under the object store's name in the Administration Console window's navigation pane.

A list of the available connection factory or destination administered objects will appear in the result pane, showing the lookup name and type of each (as well as the destination name in the case of destination administered objects).

2 Select an administered object by clicking on its name in the result pane.

3 Choose Properties from the Actions menu.

The Connection Factory Object Properties or Destination Object Properties dialog box will appear, similar to the Add Connection Factory Object (Figure 2–13) or Add Destination Object (Figure 2–14) dialog. You can use this dialog box to change the selected object's configuration attributes. Note, however, that you cannot change the object's lookup name; the only way to do this is the delete the object and then add a new administered object with the desired lookup name.

4 Click OK to accept the new attribute values and dismiss the dialog box.

Deleting an Administered Object

Deleting an administered object removes it permanently from the object store to which it belongs.

▼ To Delete an Administered Object

1 Select Connection Factories or Destinations under the object store's name in the Administration Console window's navigation pane.

A list of the available connection factory or destination administered objects will appear in the result pane, showing the lookup name and type of each (as well as the destination name in the case of destination administered objects).

2 Select an administered object by clicking on its name in the result pane.

3 Choose Delete from the Edit menu.

A confirmation dialog box will appear, asking you to confirm that you wish to proceed with the operation.

4 Click Yes to confirm the operation and dismiss the confirmation dialog.

For this exercise, do not delete the administered objects `MyQueue` or `MyQueueConnectionFactory` that you created earlier; instead, click No to dismiss the confirmation dialog without performing the delete operation.

Running the Sample Application

The sample application `HelloWorldMessageJNDI` is provided for use with this tutorial. It uses the physical destination and administered objects that you created:

- A queue physical destination named `MyQueueDest`
- A queue connection factory administered object with JNDI lookup name `MyQueueConnectionFactory`
- A queue administered object with JNDI lookup name `MyQueue`

The code creates a simple queue sender and receiver, and sends and receives a `Hello World` message.

Before running the application, open the source file `HelloWorldMessageJNDI.java` and read through the code. The program is short and amply documented; you should have little trouble understanding how it works.

▼ To Run the Sample Application

- 1 **Make the directory containing the `HelloWorldMessageJNDI` application your current directory, using one of the following commands (depending on the platform you're using):**

- On Solaris:

```
cd /usr/demo/imq/helloworld/helloworldmessagejndi
```

- On Linux:

```
cd /opt/sun/mq/examples/helloworld/helloworldmessagejndi
```

- On Windows:

```
cd IMQ_HOME\demo\helloworld\helloworldmessagejndi
```

You should find the file `HelloWorldMessageJNDI.class` present. (If you make changes to the application, you must recompile it using the procedure for compiling a client application given in the *Message Queue Developer's Guide for Java Clients*.)

- 2 **Set the `CLASSPATH` variable to include the current directory containing the file `HelloWorldMessageJNDI.class`, as well as the following `.jar` files that are included in the Message Queue product:**

```
jms.jar  
imq.jar  
jndi.jar  
fscontext.jar
```

See the *Message Queue Developer's Guide for Java Clients* for information on setting the CLASSPATH variable.

Note – The file `jndi.jar` is bundled with JDK 1.4. You need not add this file to your CLASSPATH unless you are using an earlier version of the JDK.

3 Run the HelloWorldMessageJNDI application by executing one of the following commands (depending on the platform you're using):

- On Solaris or Linux:

```
% java HelloWorldMessageJNDI file:///tmp
```

- On Windows:

```
java HelloWorldMessageJNDI
```

If the application runs successfully, you should see the output shown in [Example 2–1](#).

Example 2–1 Output from Sample Application

```
java HelloWorldMessageJNDI
Using file:///C:/Temp for Context.PROVIDER_URL

Looking up Queue Connection Factory object with lookup name:
MyQueueConnectionFactory
Queue Connection Factory object found.
Looking up Queue object with lookup name: MyQueue
Queue object found.

Creating connection to broker.
Connection to broker created.

Publishing a message to Queue: MyQueueDest
Received the following message: Hello World
```



PART II

Administrative Tasks

- Chapter 3, “Starting Brokers and Clients”
- Chapter 4, “Configuring a Broker”
- Chapter 5, “Managing a Broker ”
- Chapter 6, “Configuring and Managing Connection Services”
- Chapter 7, “Managing Message Delivery”
- Chapter 8, “Configuring Persistence Services”
- Chapter 11, “Managing Administered Objects”
- Chapter 10, “Configuring and Managing Broker Clusters”
- Chapter 9, “Configuring and Managing Security Services”
- Chapter 12, “Monitoring Broker Operations”
- Chapter 13, “Analyzing and Tuning a Message Service”
- Chapter 14, “Troubleshooting”

Starting Brokers and Clients

After installing Sun Java™ System Message Queue and performing some preparatory steps, you can begin starting brokers and clients. A broker's configuration is governed by a set of configuration files, which can be overridden by command line options passed to the Broker utility (`imqbrokerd`); see [Chapter 4, “Configuring a Broker,”](#) for more information.

This chapter contains the following sections:

- “Preparing System Resources” on page 67
- “Starting Brokers” on page 68
- “Deleting a Broker Instance” on page 74
- “Starting Clients” on page 74

Preparing System Resources

Before starting a broker, there are two preliminary system-level tasks to perform: synchronizing system clocks and (on the Solaris or Linux platform) setting the file descriptor limit. The following sections describe these tasks.

Synchronizing System Clocks

Before starting any brokers or clients, it is important to synchronize the clocks on all hosts that will interact with the Message Queue system. Synchronization is particularly crucial if you are using message expiration (time-to-live). Time stamps from clocks that are not synchronized could prevent message expiration from working as expected and prevent the delivery of messages. Synchronization is also crucial for broker clusters.

Configure your systems to run a time synchronization protocol, such as Simple Network Time Protocol (SNTP). Time synchronization is generally supported by the `xntpd` daemon on Solaris

and Linux, and by the W32Time service on Windows. (See your operating system documentation for information about configuring this service.) After the broker is running, avoid setting the system clock backward.

Setting the File Descriptor Limit

On the Solaris and Linux platforms, the shell in which a client or broker is running places a soft limit on the number of file descriptors that a process can use. In Message Queue, each connection a client makes, or a broker accepts, uses one of these file descriptors. Each physical destination that has persistent messages also uses a file descriptor.

As a result, the file descriptor limit constrains the number of connections a broker or client can have. By default, the maximum is 256 connections on Solaris or 1024 on Linux. (In practice, the connection limit is actually lower than this because of the use of file descriptors for persistent data storage.) If you need more connections than this, you must raise the file descriptor limit in each shell in which a client or broker will be executing. For information on how to do this, see the `man` page for the `ulimit` command.

Starting Brokers

You can start a broker either interactively, using the Message Queue command line utilities or the Windows Start menu, or by arranging for it to start automatically at system startup. The following sections describe how.

Starting Brokers Interactively

You can start a broker interactively from the command line, using the Broker utility (`imqbrokerd`). (Alternatively, on Windows, you can start a broker from the Start menu.) You *cannot* use the Administration Console (`imqadmin`) or the Command utility (`imqcmd`) to start a broker; the broker must already be running before you can use these tools.

On the Solaris and Linux platforms, a broker instance must always be started by the same user who initially started it. Each broker instance has its own set of configuration properties and file-based persistent data store. When the broker instance first starts, Message Queue uses the user's file creation mode mask (`umask`) to set permissions on directories containing the configuration information and persistent data for that broker instance.

A broker instance has the instance name `imqbroker` by default. To start a broker from the command line with this name and the default configuration, simply use the command

```
imqbrokerd
```

This starts a broker instance named `imqbroker` on the local machine, with the Port Mapper at the default port of 7676 (see [“Port Mapper” on page 95](#)).

To specify an instance name other than the default, use the `-name` option to the `imqbrokerd` command. The following command starts a broker with the instance name `myBroker`:

```
imqbrokerd -name myBroker
```

Other options are available on the `imqbrokerd` command line to control various aspects of the broker's operation. See [“Broker Utility” on page 278](#) for complete information on the syntax, subcommands, and options of the `imqbrokerd` command. For a quick summary of this information, enter the following command:

```
imqbrokerd -help
```

For example, the following command uses the `-tty` option to send errors and warnings to the command window (standard output):

```
imqbrokerd -name myBroker -tty
```

You can also use the `-D` option on the command line to override the values of properties specified in the broker's instance configuration file (`config.properties`). The instance configuration file is described under [“Modifying Configuration Files” on page 78](#). The following example sets a broker's `imq.jms.max_threads` property, raising the maximum number of threads available to the `jms` connection service to 2000:

```
imqbrokerd -name myBroker -Dimq.jms.max_threads=2000
```

Starting Brokers Automatically

Instead of starting a broker explicitly from the command line, you can set it up to start automatically at system startup. How you do this depends on the platform (Solaris, Linux, or Windows) on which you are running the broker:

- [“Automatic Broker Startup on the Solaris Platforms” on page 69](#)
- [“Automatic Broker Startup on the Linux Platform” on page 71](#)
- [“Automatic Broker Startup on Windows” on page 71](#)

Automatic Broker Startup on the Solaris Platforms

The method for enabling automatic startup on the Solaris 10 platforms is different from that for Solaris 9. Both are described below.

Automatic Broker Startup on the Solaris 9 Platform

On Solaris 9 operating system, scripts that enable automatic startup are placed in the `/etc/rc*` directory tree during Message Queue installation. To enable the use of these scripts, you must edit the configuration file `/etc/imq/imqbrokerd.conf` as follows:

- To start the broker automatically at system startup, set the `AUTOSTART` property to `YES`.

- To have the broker restart automatically after an abnormal exit, set the `RESTART` property to `YES`.
- To set startup command line arguments for the broker, specify one or more values for the `ARGS` property.

To disable automatic broker startup at system startup, edit the configuration file `/etc/immq/immqbrokerd.conf` and set the `AUTOSTART` property to `NO`.

Automatic Broker Startup on the Solaris 10 Platform

Rather than using an `rc` file to implement automatic broker startup when a computer reboots, the following procedure makes use of the Solaris 10 Service Management Facility (SMF).

For more information on using the Service Management Facility, please refer to Solaris 10 documentation.

▼ To Implement Automatic Broker Startup on Solaris 10 OS

1 Import the `mqbroker` service into the SMF repository.

```
# svccfg import /var/svc/manifest/application/sun/mq/mqbroker.xml
```

2 Verify that the import was successful by checking the state of the `mqbroker` service.

```
# svcs mqbroker
```

Output resembles the following:

```
STATE TIME FMRI
disabled 16:22:50 svc:/application/sun/mq/mqbroker:default
```

The service is initially shown as disabled.

3 Enable the `mqbroker` service.

```
# svcadm enable svc:/application/sun/mq/mqbroker:default
```

Enabling the `mqbroker` service will start the `immqbrokerd` process. A reboot will subsequently restart the broker.

4 Configure the `mqbroker` service to pass any desired arguments to the `immqbrokerd` command.

The `options/broker_args` property is used to pass arguments to `immqbrokerd`. For example to add `-loglevel DEBUGHIGH`, do the following:

```
# svccfg
svc:> select svc:/application/sun/mq/mqbroker
svc:/application/sun/mq/mqbroker> setprop options/broker_args="-loglevel DEBUGHIGH"
svc:/application/sun/mq/mqbroker> exit
```

▼ To Disable Automatic Broker Startup on Solaris 10 OS

- **Disable the `mqbroker` service.**

```
# svcadm disable svc:/application/sun/mq/mqbroker:default
```

A subsequent reboot will not restart the broker.

Automatic Broker Startup on the Linux Platform

On Linux systems, scripts that enable automatic startup are placed in the `/etc/rc*` directory tree during Message Queue installation. To enable the use of these scripts, you must edit the configuration file `/etc/opt/sun/mq/imqbrokerd.conf` as follows:

- To start the broker automatically at system startup, set the `AUTOSTART` property to `YES`.
- To have the broker restart automatically after an abnormal exit, set the `RESTART` property to `YES`.
- To set startup command line arguments for the broker, specify one or more values for the `ARGS` property.

To disable automatic broker startup at system startup, edit the configuration file `/etc/opt/sun/mq/imqbrokerd.conf` and set the `AUTOSTART` property to `NO`.

Automatic Broker Startup on Windows

To start a broker automatically at Windows system startup, you must define the broker as a Windows service. The broker will then start at system startup time and run in the background until system shutdown. Consequently, you will not need to use the Message Queue Broker utility (`imqbrokerd`) unless you want to start an additional broker.

A system can have no more than one broker running as a Windows service. The Windows Task Manager lists such a broker as two executable processes:

- The native Windows service wrapper, `imqbrokersvc.exe`
- The Java runtime that is running the broker

You can install a broker as a service when you install Message Queue on a Windows system. After installation, you can use the Service Administrator utility (`imqsvcadm`) to perform the following operations:

- Add a broker as a Windows service
- Determine the startup options for the broker service
- Disable a broker from running as a Windows service

To pass startup options to the broker, use the `-args` option to the `imqsvcadm` command. This works the same way as the `imqbrokerd` command's `-D` option, as described under [“Starting Brokers” on page 68](#). Use the Command utility (`imqcmd`) to control broker operations as usual.

See “[Service Administrator Utility](#)” on page 297 for complete information on the syntax, subcommands, and options of the `imqsvcadmin` command.

Reconfiguring the Broker Service

The procedure for reconfiguring a broker installed as a Windows service is as follows:

▼ To Reconfigure a Broker Running as a Windows Service

1 Stop the service:

- a. From the Settings submenu of the Windows Start menu, choose Control Panel.
- b. Open the Administrative Tools control panel.
- c. Run the Services tool by selecting its icon and choosing Open from the File menu or the pop-up context menu, or simply by double-clicking the icon.
- d. Under Services (Local), select the Message Queue Broker service and choose Properties from the Action menu.

Alternatively, you can right-click on Message Queue Broker and choose Properties from the pop-up context menu, or simply double-click on Message Queue Broker. In either case, the Message Queue Broker Properties dialog box will appear.

- e. Under the General tab in the Properties dialog, click Stop to stop the broker service.

2 Remove the service.

On the command line, enter the command

```
imqsvcadmin remove
```

3 Reinstall the service, specifying different broker startup options with the `-args` option or different Java version arguments with the `-vmargs` option.

For example, to change the service’s host name and port number to `broker1` and `7878`, you could use the command

```
imqsvcadmin install -args "-name broker1 -port 7878"
```

Using an Alternative Java Runtime

You can use either the `imqsvcadmin` command’s `-javahome` or `-jrehome` option to specify the location of an alternative Java runtime. (You can also specify these options in the Start Parameters field under the General tab in the service’s Properties dialog window.)

Note – The Start Parameters field treats the backslash character (\) as an escape character, so you must type it twice when using it as a path delimiter: for example,

```
-javahome c:\\j2sdk1.4.0
```

Displaying Broker Service Startup Options

To determine the startup options for the broker service, use the `imqsvcadmin query` command, as shown in [Example 3-1](#).

EXAMPLE 3-1 Displaying Broker Service Startup Options

```
imqsvcadmin query

Service Message Queue Broker is installed.
Display Name: Message Queue Broker
Start Type: Automatic
Binary location: C:\Sun\MessageQueue\bin\imqbrokersvc.exe
JavaHome: c:\j2sdk1.4.0
Broker Args: -name broker1 -port 7878
```

Disabling a Broker From Running as a Windows Service

To disable a broker from running as a Windows service, use the command

```
imqcmd shutdown bkr
```

to shut down the broker, followed by

```
imqsvcadmin remove
```

to remove the service.

Alternatively, you can use the Windows Services tool, reached via the Administrative Tools control panel, to stop and remove the broker service.

Restart your computer after disabling the broker service.

Troubleshooting Service Startup Problems

If you get an error when you try to start a broker as a Windows service, you can view error events that were logged:

▼ To See Logged Service Error Events

- 1 Open the Windows Administrative Tools control panel.
- 2 Start the Event Viewer tool.
- 3 Select the Application event log.
- 4 Choose Refresh from the Action menu to display any error events.

Deleting a Broker Instance

To delete a broker instance, use the `imqbrokerd` command with the `-remove` option:

```
imqbrokerd [options...] -remove instance
```

For example, if the name of the broker is `myBroker`, the command would be

```
imqbrokerd -name myBroker -remove instance
```

The command deletes the entire instance directory for the specified broker.

See [“Broker Utility” on page 278](#) for complete information on the syntax, subcommands, and options of the `imqbrokerd` command. For a quick summary of this information, enter the command

```
imqbrokerd -help
```

Starting Clients

Before starting a client application, obtain information from the application developer about how to set up the system. If you are starting Java client applications, you must set the `CLASSPATH` variable appropriately and make sure you have the correct `.jar` files installed. The *Message Queue Developer’s Guide for Java Clients* contains information about generic steps for setting up the system, but your developer may have additional information to provide.

To start a Java client application, use the following command line format:

```
java clientAppName
```

To start a C client application, use the format supplied by the application developer (see [“Building and Running C Clients” in *Sun Java System Message Queue 4.2 Developer’s Guide for C Clients*](#)).

The application's documentation should provide information on attribute values that the application sets; you may want to override some of these from the command line. You may also want to specify attributes on the command line for any Java client that uses a Java Naming and Directory Interface (JNDI) lookup to find its connection factory. If the lookup returns a connection factory that is older than the application, the connection factory may lack support for more recent attributes. In such cases, Message Queue sets those attributes to default values; if necessary, you can use the command line to override these default values.

To specify attribute values from the command line for a Java application, use the following syntax:

```
java [ [-Dattribute=value] ... ] clientAppName
```

The value for *attribute* must be a connection factory administered object attribute, as described in [Chapter 18, “Administered Object Attribute Reference.”](#) If there is a space in the value, put quotation marks around the

```
attribute=value
```

part of the command line.

The following example starts a client application named MyMQClient, connecting to a broker on the host OtherHost at port 7677:

```
java -DmqAddressList=mq://OtherHost:7677/jms MyMQClient
```

The host name and port specified on the command line override any others set by the application itself.

In some cases, you cannot use the command line to specify attribute values. An administrator can set an administered object to allow read access only, or an application developer can code the client application to do so. Communication with the application developer is necessary to understand the best way to start the client program.

Configuring a Broker

A broker's configuration is governed by a set of configuration files and by the options passed to the `imqbrokerd` command at startup. This chapter describes the available configuration properties and how to use them to configure a broker.

The chapter contains the following sections:

- “Broker Services” on page 77
- “Setting Broker Configuration Properties” on page 78

For full reference information about broker configuration properties, see [Chapter 16, “Broker Properties Reference”](#)

Broker Services

Broker configuration properties are logically divided into categories that depend on the services or broker components they affect:

- *Connection services* manage the physical connections between a broker and its clients that provide transport for incoming and outgoing messages. For a discussion of properties associated with connection services, see [“Configuring Connection Services” on page 93](#)
- *Message delivery services* route and deliver JMS payload messages, as well as control messages used by the message service to support reliable delivery. For a discussion of properties associated with message delivery services, including physical destinations, see [Chapter 7, “Managing Message Delivery”](#)
- *Persistence services* manage the writing and retrieval of data, such as messages and state information, to and from persistent storage. For a discussion of properties associated with persistence services, see [Chapter 8, “Configuring Persistence Services”](#)
- *Security services* authenticate users connecting to the broker and authorize their actions. For a discussion of properties associated with authentication and authorization services, as well as encryption configuration, see [Chapter 9, “Configuring and Managing Security Services”](#)

- *Clustering services* support the grouping of brokers into a cluster to achieve scalability and availability. For a discussion of properties associated with broker clusters, see [Chapter 10, “Configuring and Managing Broker Clusters”](#)
- *Monitoring services* generate metric and diagnostic information about the broker’s performance. For a discussion of properties associated with monitoring and managing a broker, see [Chapter 12, “Monitoring Broker Operations”](#)

Setting Broker Configuration Properties

You can specify a broker’s configuration properties in either of two ways:

- Edit the broker’s configuration file.
- Supply the property values directly from the command line.

The following sections describe these two methods of configuring a broker.

Modifying Configuration Files

Broker configuration files contain property settings for configuring a broker. They are kept in a directory whose location depends on the operating system platform you are using; see [Appendix A, “Platform-Specific Locations of Message Queue Data,”](#) for details. Message Queue maintains the following broker configuration files:

- A *default configuration file* (`default.properties`) that is loaded on startup. This file is not editable, but you can read it to determine default settings and find the exact names of properties you want to change.
- An *installation configuration file* (`install.properties`) containing any properties specified when Message Queue was installed. This file cannot be edited after installation.
- A separate *instance configuration file* (`config.properties`) for each individual broker instance.

In addition, if you connect broker instances in a cluster, you may need to use a *cluster configuration file* (`cluster.properties`) to specify configuration information for the cluster; see [“Cluster Configuration Properties” on page 321](#) for more information.

Also, Message Queue makes use of an environment configuration file, `imqenv.conf`, which is used to provide the locations of external files needed by Message Queue, such as the default Java SE location and the locations of database drivers, JAAS login modules, and so forth.

At startup, the broker merges property values from the various configuration files. As shown in [Figure 4–1](#), the files form a hierarchy in which values specified in the instance configuration file override those in the installation configuration file, which in turn override those in the default

configuration file. At the top of the hierarchy, you can manually override any property values specified in the configuration files by using command line options to the `mqbrokerd` command.

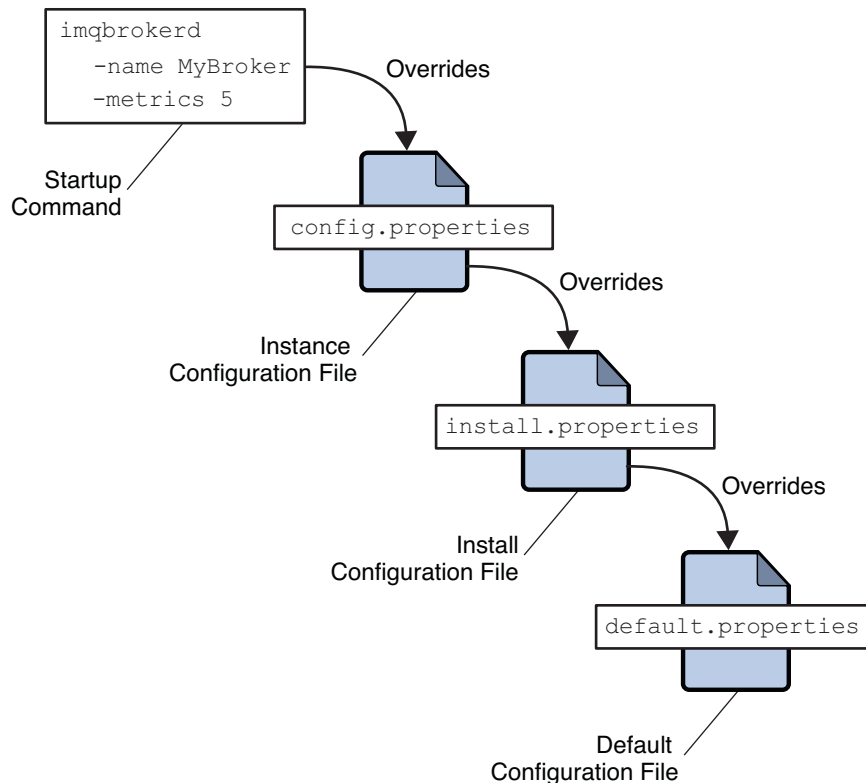


FIGURE 4-1 Broker Configuration Files

The first time you run a broker, an instance configuration file is created containing configuration properties for that particular broker instance. The instance configuration file is named `config.properties` and is located in a directory identified by the name of the broker instance to which it belongs:

```
.../instances/instanceName/props/config.properties
```

(See [Appendix A, “Platform-Specific Locations of Message Queue Data,”](#) for the location of the `instances` directory.) If the file does not yet exist, you must use the `-name` option when starting the broker (see [“Broker Utility” on page 278](#)) to specify an instance name that Message Queue can use to create the file.

Note – The `instances/instanceName` directory and the instance configuration file are owned by the user who initially started the corresponding broker instance by using the `imqbrokerd -name brokerName` option. The broker instance must always be restarted by that same user.

The instance configuration file is maintained by the broker instance and is modified when you make configuration changes using Message Queue administration utilities. You can also edit an instance configuration file by hand. To do so, you must be the owner of the `instances/instanceName` directory or log in as the root user to change the directory's access privileges.

The broker reads its instance configuration file only at startup. To effect any changes to the broker's configuration, you must shut down the broker and then restart it. Property definitions in the `config.properties` file (or any configuration file) use the following syntax:

```
propertyName=value [ [,value1] ... ]
```

For example, the following entry specifies that the broker will hold up to 50,000 messages in memory and persistent storage before rejecting additional messages:

```
imq.system.max_count=50000
```

The following entry specifies that a new log file will be created once a day (every 86,400 seconds):

```
imq.log.file.rolloversecs=86400
```

See [“Broker Services” on page 77](#) and [Chapter 16, “Broker Properties Reference,”](#) for information on the available broker configuration properties and their default values.

Setting Configuration Properties from the Command Line

You can enter broker configuration properties from the command line when you start a broker, or afterward.

At startup time, you use the Broker utility (`imqbrokerd`) to start a broker instance. Using the command's `-D` option, you can specify any broker configuration property and its value; see [“Starting Brokers” on page 68](#) and [“Broker Utility” on page 278](#) for more information. If you start the broker as a Windows service, using the Service Administrator utility (`imqsvcadm`), you use the `-args` option to specify startup configuration properties; see [“Service Administrator Utility” on page 297](#).

You can also change certain broker configuration properties while a broker is running. To modify the configuration of a running broker, you use the Command utility's `imqcmd update bkr` command; see [“Updating Broker Properties” on page 89](#) and [“Broker Management” on page 285](#).

Managing a Broker

This chapter explains how to use the Message Queue Command utility (`imqcmd`) to manage a broker. The chapter has the following sections:

- “[Command Utility Preliminaries](#)” on page 84
- “[Using the Command Utility](#)” on page 84
- “[Managing Brokers](#)” on page 87

This chapter does not cover all topics related to managing a broker. Additional topics are covered in the following separate chapters:

- For information on configuring and managing connection services, see [Chapter 6](#), “[Configuring and Managing Connection Services](#).”
- For information on managing message delivery services, including how to create, display, update, and destroy physical destinations, see [Chapter 7](#), “[Managing Message Delivery](#).”
- For information on configuring and managing persistence services, for both flat-file and JDBC-based data stores, see [Chapter 8](#), “[Configuring Persistence Services](#).”
- For information about setting up security for the broker, such as user authentication, access control, encryption, and password files, see [Chapter 9](#), “[Configuring and Managing Security Services](#).”
- For information on configuring and managing clustering services, for both standard and high-availability broker clusters, see [Chapter 10](#), “[Configuring and Managing Broker Clusters](#).”
- For information about monitoring a broker, see [Chapter 12](#), “[Monitoring Broker Operations](#).”

Command Utility Preliminaries

Before using the Command utility to manage a broker, you must do the following:

- Start the broker using the `imqbrokerd` command. You cannot use the Command utility subcommands `l` until a broker is running.
- Determine whether you want to set up a Message Queue administrative user or use the default account. You must specify a user name and password to use all Command utility subcommands (except to display command help and version information).

When you install Message Queue, a default flat-file user repository is installed. The repository is shipped with two default entries: an administrative user and a guest user. If you are testing Message Queue, you can use the default user name and password (`admin/admin`) to run the Command utility.

If you are setting up a production system, you must set up authentication and authorization for administrative users. See [Chapter 9, “Configuring and Managing Security Services,”](#) for information on setting up a file-based user repository or configuring the use of an LDAP directory server. In a production environment, it is a good security practice to use a nondefault user name and password.

- If you want to use a secure connection to the broker, set up and enable the `ssladmin` service on the target broker instance. For more information, see [“Message Encryption” on page 159](#).

Using the Command Utility

The Message Queue Command utility (`imqcmd`) enables you to manage the broker and its services interactively from the command line. See [“Command Utility” on page 282](#) for general reference information about the syntax, subcommands, and options of the `imqcmd` command, and [Chapter 16, “Broker Properties Reference,”](#) for specific information on the configuration properties used to specify broker behavior.

Specifying the User Name and Password

Because each `imqcmd` subcommand is authenticated against the user repository, it requires a user name and password. The only exceptions are commands that use the `-h` or `-H` option to display help, and those that use the `-v` option to display the product version.

Use the `-u` option to specify an administrative user name. For example, the following command displays information about the default broker:

```
imqcmd query bkr -u admin
```

If you omit the user name, the command will prompt you for it.

Note – For simplicity, the examples in this chapter use the default user name `admin` as the argument to the `-u` option. In a real-life production environment, you would use a custom user name.

Specify the password using one of the following methods:

- Create a *password file* and enter the password into that file as the value of the `imq.imqcmd.password` property. On the command line, use the `-passfile` option to provide the name of the password file.
- Let the `imqcmd` command prompt you for the password.

Note – In previous versions of Message Queue, you could use the `-p` option to specify a password on the `imqcmd` command line. As of Message Queue 4.0, this option is deprecated and is no longer supported; you must instead use one of the methods listed above.

Specifying the Broker Name and Port

Most `imqcmd` subcommands use the `-b` option to specify the host name and port number of the broker to which the command applies:

`-b hostName:portNumber`

If no broker is specified, the command applies by default to a broker running on the local host (`localhost`) at port number 7676. To issue a command to a broker that is running on a remote host, listening on a nondefault port, or both, you must use the `-b` option to specify the host and port explicitly.

Displaying the Product Version

To display the Message Queue product version, use the `-v` option. For example:

```
imqcmd -v
```

If you enter an `imqcmd` command line containing the `-v` option in addition to a subcommand or other options, the Command utility processes only the `-v` option. All other items on the command line are ignored.

Displaying Help

To display help on the `imqcmd` command, use the `-h` or `-H` option, and do not use a subcommand. You cannot get help about specific subcommands.

For example, the following command displays help about `imqcmd`:

```
imqcmd -H
```

If you enter an `imqcmd` command line containing the `-h` or `-H` option in addition to a subcommand or other options, the Command utility processes only the `-h` or `-H` option. All other items on the command line are ignored.

Examples

The examples in this section illustrate how to use the `imqcmd` command.

The following example lists the properties of the broker running on host `localhost` at port `7676`, so the `-b` option is unnecessary:

```
imqcmd query bkr -u admin
```

The command uses the default administrative user name (`admin`) and omits the password, so that the command will prompt for it.

The following example lists the properties of the broker running on the host `myserver` at port `1564`. The user name is `aladdin`:

```
imqcmd query bkr -b myserver:1564 -u aladdin
```

(For this command to work, the user repository would need to be updated to add the user name `aladdin` to the `admin` group.)

The following example lists the properties of the broker running on `localhost` at port `7676`. The initial timeout for the command is set to 20 seconds and the number of retries after timeout is set to 7. The user's password is in a password file called `myPassfile`, located in the current directory at the time the command is invoked.

```
imqcmd query bkr -u admin -passfile myPassfile -rtm 20 -rtr 7
```

For a secure connection to the broker, these examples could include the `-secure` option. This option causes the Command utility to use the `ssladmin` service if that service has been configured and started.

Managing Brokers

This section describes how to use Command utility subcommands to perform the following broker management tasks:

- [“Shutting Down and Restarting a Broker” on page 87](#)
- [“Quiescing a Broker” on page 88](#)
- [“Pausing and Resuming a Broker” on page 89](#)
- [“Updating Broker Properties” on page 89](#)
- [“Viewing Broker Information” on page 91](#)

In addition to using the subcommands described in the following sections, `imqcmd` allows you to set system properties using the `-D` option. This is useful for setting or overriding connection factory properties or connection-related Java system properties.

For example, the following command changes the default value of `imqSSLIsHostTrusted`:

```
imqcmd list svc -secure -DimqSSLIsHostTrusted=true
```

The following command specifies the password file and the password used for the SSL trust store that is used by the `imqcmd` command:

```
imqcmd list svc -secure -Djavax.net.ssl.trustStore=/tmp/MyTruststore
-Djavax.net.ssl.trustStorePassword=MyTrustword
```

Shutting Down and Restarting a Broker

The subcommand `imqcmd shutdown bkr` shuts down a broker:

```
imqcmd shutdown bkr [-b hostName:portNumber]
                    [-time nSeconds]
                    [-nofailover]
```

The broker stops accepting new connections and messages, completes delivery of existing messages, and terminates the broker process.

The `-time` option, if present, specifies the interval, in seconds, to wait before shutting down the broker. For example, the following command delays 90 seconds and then shuts down the broker running on host `wolfgang` at port 1756:

```
imqcmd shutdown bkr -b wolfgang:1756 -time 90 -u admin
```

The broker will not block, but will return immediately from the delayed shutdown request. During the shutdown interval, the broker will not accept any new `jms` connections; `admin` connections will be accepted, and existing `jms` connections will continue to operate. If the broker belongs to a high-availability cluster, it will not attempt to take over for any other broker during the shutdown interval.

If the broker is part of a high-availability cluster (see “High-Availability Clusters” in *Sun Java System Message Queue 4.2 Technical Overview* “High-Availability Clusters” in *Sun Java System Message Queue 4.2 Technical Overview*), another broker in the cluster will ordinarily attempt to take over its persistent data on shutdown; the `-nofailover` option to the `imqcmd shutdown bkr` subcommand suppresses this behavior. Conversely, you can use the `imqcmd takeover bkr` subcommand to force such a takeover manually (for instance, if the takeover broker were to fail before completing the takeover process); see “Preventing or Forcing Broker Failover” on page 186 for more information.

Note – The `imqcmd takeover bkr` subcommand is intended only for use in failed-takeover situations. You should use it only as a last resort, and not as a general way of forcibly taking over a running broker.

To shut down and restart a broker, use the subcommand `imqcmd restart bkr`:

```
imqcmd restart bkr [-b hostName:portNumber]
```

This shuts down the broker and then restarts it using the same options that were specified when it was first started. To choose different options, shut down the broker with `imqcmd shutdown bkr` and then start it again with the Broker utility (`imqbrokerd`), specifying the options you want.

Quiescing a Broker

The subcommand `imqcmd quiesce bkr` *quiesces* a broker, causing it to refuse any new client connections while continuing to service old ones:

```
imqcmd quiesce bkr [-b hostName:portNumber]
```

If the broker is part of a high-availability cluster, this allows its operations to wind down normally without triggering a takeover by another broker, for instance in preparation for shutting it down administratively for upgrade or similar purposes. For example, the following command quiesces the broker running on host `hastings` at port `1066`:

```
imqcmd quiesce bkr -b hastings:1066 -u admin
```

To reverse the process and return the broker to normal operation, use the `imqcmd unquiesce bkr` subcommand:

```
imqcmd unquiesce bkr [-b hostName:portNumber]
```

For example, the following command unquiesces the broker that was quiesced in the preceding example:

```
imqcmd unquiesce bkr -b hastings:1066 -u admin
```


Pausing and Resuming a Broker

The subcommand `imqcmd pause bkr` pauses a broker, suspending its connection service threads and causing it to stop listening on the connection ports:

```
imqcmd pause bkr [-b hostName:portNumber]
```

For example, the following command pauses the broker running on host `myhost` at port 1588:

```
imqcmd pause bkr -b myhost:1588 -u admin
```

Because its connection service threads are suspended, a paused broker is unable to accept new connections, receive messages, or dispatch messages. The `admin` connection service is *not* suspended, allowing you to continue performing administrative tasks needed to regulate the flow of messages to the broker. Pausing a broker also does not suspend the `cluster` connection service; however, since message delivery within a cluster depends on the delivery functions performed by the different brokers in the cluster, pausing a broker in a cluster may result in a slowing of some message traffic.

You can also pause individual connection services and physical destinations. For more information, see [“Pausing and Resuming a Connection Service” on page 97](#) and [“Pausing and Resuming a Physical Destination” on page 110](#).

The subcommand `imqcmd resume bkr` reactivates a broker’s service threads, causing it to resume listening on the ports:

```
imqcmd resume bkr [-b hostName:portNumber]
```

For example, the following command resumes the default broker (host `localhost` at port 7676):

```
imqcmd resume bkr -u admin
```

Updating Broker Properties

The subcommand `imqcmd update bkr` can be used to change the values of a subset of broker properties for the default broker (or for the broker at a specified host and port):

```
imqcmd update bkr [-b hostName:portNumber
                  -o property1=value1 [ [-o property2=value2] ... ]
```

For example, the following command turns off the auto-creation of queue destinations for the default broker:

```
imqcmd update bkr -o imq.autocreate.queue=false -u admin
```

You can use `imqcmd update bkr` to update any of the following broker properties:

```
imq.autocreate.queue  
imq.autocreate.topic  
imq.autocreate.queue.maxNumActiveConsumers  
imq.autocreate.queue.maxNumBackupConsumers  
imq.cluster.url  
imq.destination.DMQ.truncateBody  
imq.destination.logDeadMsgs  
imq.log.level  
imq.log.file.rolloversecs  
imq.log.file.rolloverbytes  
imq.system.max_count  
imq.system.max_size  
imq.message.max_size  
imq.portmapper.port
```

See [Chapter 16, “Broker Properties Reference,”](#) for detailed information about these properties.

Viewing Broker Information

To display information about a single broker, use the `imqcmd query bkr` subcommand:

```
imqcmd query bkr -b hostName:portNumber
```

This lists the current settings of the broker’s properties, as shown in [Example 5–1](#).

EXAMPLE 5–1 Broker Information Listing

Querying the broker specified by:

Host Primary Port

localhost 7676

Version	4.2
Instance Name	imqbroker
Broker ID	mybroker
Primary Port	7676
Broker is Embedded	false
Instance Configuration/Data Root Directory	/var/imq
Current Number of Messages in System	0
Current Total Message Bytes in System	0
Current Number of Messages in Dead Message Queue	0
Current Total Message Bytes in Dead Message Queue	0
Log Dead Messages	true
Truncate Message Body in Dead Message Queue	false
Max Number of Messages in System	unlimited (-1)
Max Total Message Bytes in System	unlimited (-1)
Max Message Size	70m
Auto Create Queues	true
Auto Create Topics	true
Auto Created Queue Max Number of Active Consumers	1
Auto Created Queue Max Number of Backup Consumers	0
Cluster ID	myClusterID
Cluster Is Highly Available	true
Cluster Broker List (active)	
Cluster Broker List (configured)	
Cluster Master Broker	
Cluster URL	
Log Level	INFO
Log Rollover Interval (seconds)	604800
Log Rollover Size (bytes)	unlimited (-1)

Chapter 5 • Managing a Broker

Successfully queried the broker.

The `imqcmd metrics bkr` subcommand displays detailed metric information about a broker's operation:

```
imqcmd metrics bkr [-b hostName:portNumber]  
                  [-m metricType]  
                  [-int interval]  
                  [-msp numSamples]
```

The `-m` option specifies the type of metric information to display:

- `ttl` (*default*): Messages and packets flowing into and out of the broker
- `rts`: Rate of flow of messages and packets into and out of the broker per second
- `cxn`: Connections, virtual memory heap, and threads

The `-int` and `-msp` options specify, respectively, the interval (in seconds) at which to display the metrics and the number of samples to display in the output. The default values are 5 seconds and an unlimited number of samples.

For example, the following command displays the rate of message flow into and out of the default broker (host `localhost` at port `7676`) at 10-second intervals:

```
imqcmd metrics bkr -m rts -int 10 -u admin
```

[Example 5–2](#) shows an example of the resulting output.

EXAMPLE 5–2 Broker Metrics Listing

Msgs/sec		Msg Bytes/sec		Pkts/sec		Pkt Bytes/sec	
In	Out	In	Out	In	Out	In	Out
0	0	27	56	0	0	38	66
10	0	7365	56	10	10	7457	1132
0	0	27	56	0	0	38	73
0	10	27	7402	10	20	1400	8459
0	0	27	56	0	0	38	73

For a more detailed description of the data gathered and reported by the broker, see [“Brokerwide Metrics” on page 356](#).

For brokers belonging to a broker cluster, the `imqcmd list bkr` subcommand displays information about the configuration of the cluster; see [“Displaying a Cluster Configuration” on page 176](#) for more information.

Configuring and Managing Connection Services

Message Queue offers various *connection services* using a variety of transport protocols for connecting both application and administrative clients to a broker. This chapter describes how to configure and manage these services and the connections they support:

- “Configuring Connection Services” on page 93
- “Managing Connection Services” on page 97
- “Managing Connections” on page 101

Configuring Connection Services

Broker configuration properties related to connection services are listed under “[Connection Properties](#)” on page 299.

[Figure 6–1](#) shows the connection services provided by the Message Queue broker.

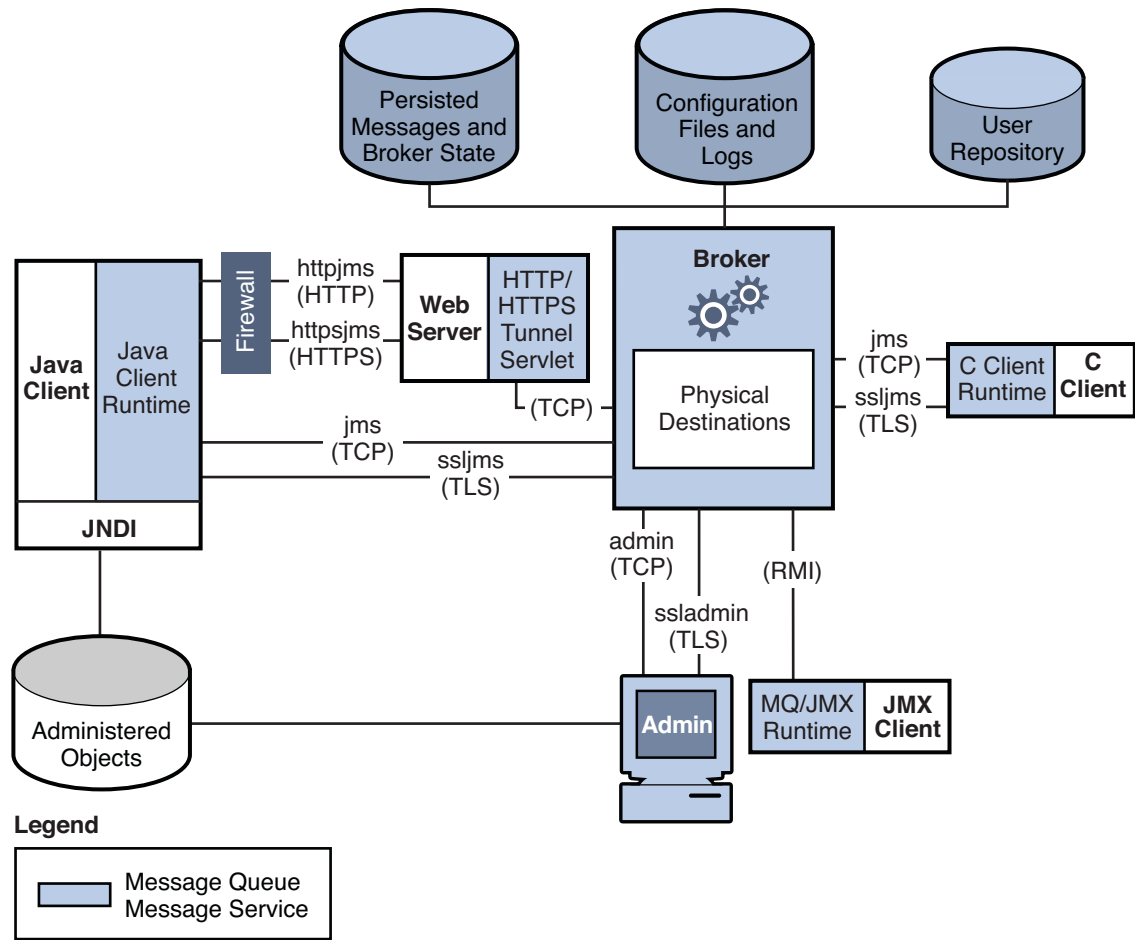


FIGURE 6-1 Message Queue Connection Services

These connection services are distinguished by two characteristics, as shown in [Table 6-1](#):

- The *service type* specifies whether the service provides JMS message delivery (NORMAL) or Message Queue administration services (ADMIN).
- The *protocol type* specifies the underlying transport protocol.

TABLE 6-1 Message Queue Connection Service Characteristics

Service Name	Service Type	Protocol Type
jms	NORMAL	TCP
ssljms	NORMAL	TLS (SSL-based security)

TABLE 6-1 Message Queue Connection Service Characteristics (Continued)

Service Name	Service Type	Protocol Type
httpjms	NORMAL	HTTP
httpsjms	NORMAL	HTTPS (SSL-based security)
admin	ADMIN	TCP
ssladmin	ADMIN	TLS (SSL-based security)

By setting a broker's `imq.service.activelist` property, you can configure it to run any or all of these connection services. The value of this property is a list of connection services to be activated when the broker is started up; if the property is not specified explicitly, the `jms` and `admin` services will be activated by default.

Each connection service also supports specific authentication and authorization features; see [“Introduction to Security Services” on page 135](#) for more information.

Note – There is also a special `cluster` connection service, used internally by the brokers within a broker cluster to exchange information about the cluster's configuration and state. This service is not intended for use by clients communicating with a broker. See [Chapter 10, “Configuring and Managing Broker Clusters,”](#) for more information about broker clusters.

Also there are two JMX connectors, `jmxrmi` and `ssljmxrmi`, that support JMX-based administration. These JMX connectors are very similar to the connection services in [Table 6-1](#), above, and are used by JMX clients to establish a connection to the broker's MBean server. For more information, see [“JMX Connection Infrastructure” on page 399](#).

Port Mapper

Each connection service is available at a particular port, specified by host name (or IP address) and port number. You can explicitly specify a static port number for a service or have the broker's *Port Mapper* assign one dynamically. The Port Mapper itself resides at the broker's *primary port*, which is normally located at the standard port number 7676. (If necessary, you can use the broker configuration property `imq.portmapper.port` to override this with a different port number.) By default, each connection service registers itself with the Port Mapper when it starts up. When a client creates a connection to the broker, the Message Queue client runtime first contacts the Port Mapper, requesting a port number for the desired connection service.

Alternatively, you can override the Port Mapper and explicitly assign a static port number to a connection service, using the `imq.serviceName.protocolType.port` configuration property (where *serviceName* and *protocolType* identify the specific connection service, as shown in [Table 6-1](#)). (Only the `jms`, `ssljms`, `admin`, and `ssladmin` connection services can be configured

this way; the `httpjms` and `httpsjms` services use different configuration properties, described in [Appendix C, “HTTP/HTTPS Support”](#)). Static ports are generally used only in special situations, however, such as in making connections through a firewall (see [“Connecting Through a Firewall” on page 169](#)), and are not recommended for general use.

Note – In cases where two or more hosts are available (such as when more than one network interface card is installed in a computer), you can use broker properties to specify which host the connection services should bind to. The `imq.hostname` property designates a single default host for all connection services; this can then be overridden, if necessary, with `imq.serviceName.protocolType.hostname` (for the `jms`, `ssljms`, `admin`, or `ssladmin` service) or `imq.portmapper.hostname` (for the Port Mapper itself).

When multiple Port Mapper requests are received concurrently, they are stored in an operating system backlog while awaiting action. The `imq.portmapper.backlog` property specifies the maximum number of such backlogged requests. When this limit is exceeded, any further requests will be rejected until the backlog is reduced.

Thread Pool Management

Each connection service is multithreaded, supporting multiple connections. The threads needed for these connections are maintained by the broker in a separate *thread pool* for each service. As threads are needed by a connection, they are added to the thread pool for the service supporting that connection.

The threading model you choose specifies whether threads are dedicated to a single connection or shared by multiple connections:

- In the *dedicated model*, each connection to the broker requires two threads: one for incoming and one for outgoing messages. This limits the number of connections that can be supported, but provides higher performance.
- In the *shared model*, connections are processed by a shared thread when sending or receiving messages. Because each connection does not require dedicated threads, this model increases the number of possible connections, but at the cost of lower performance because of the additional overhead needed for thread management.

The broker's `imq.serviceName.threadpool_model` property specifies which of the two models to use for a given connection service. This property takes either of two string values: `dedicated` or `shared`. If you don't set the property explicitly, `dedicated` is assumed by default.

You can also set the broker properties `imq.serviceName.min_threads` and `imq.serviceName.max_threads` to specify a minimum and maximum number of threads in a service's thread pool. When the number of available threads exceeds the specified minimum threshold, Message Queue will shut down threads as they become free until the minimum is reached again, thereby

saving on memory resources. Under heavy loads, the number of threads might increase until the pool's maximum number is reached; at this point, new connections are rejected until a thread becomes available.

The shared threading model uses *distributor threads* to assign threads to active connections. The broker property `imq.shared.connectionMonitor_limit` specifies the maximum number of connections that can be monitored by a single distributor thread. The smaller the value of this property, the faster threads can be assigned to connections. The `imq.ping.interval` property specifies the time interval, in seconds, at which the broker will periodically test (“ping”) a connection to verify that it is still active, allowing connection failures to be detected preemptively before an attempted message transmission fails.

Managing Connection Services

Message Queue brokers support connections from both application clients and administrative clients. See [“Configuring Connection Services” on page 93](#) for a description of the available connection services. The Command utility provides subcommands that you can use for managing both connection services as a whole and individual services; to apply a subcommand to a particular service, use the `-n` option to specify one of the names listed in the “Service Name” column of [Table 6–1](#). Subcommands are available for the following connection service management tasks:

- [“Pausing and Resuming a Connection Service” on page 97](#)
- [“Updating Connection Service Properties” on page 98](#)
- [“Viewing Connection Service Information” on page 99](#)

Pausing and Resuming a Connection Service

Pausing a connection service has the following effects:

- The broker stops accepting new client connections on the paused service. If a Message Queue client attempts to open a new connection, it will get an exception.
- All existing connections on the paused service are kept alive, but the broker suspends all message processing on such connections until the service is resumed. (For example, if a client attempts to send a message, the `send` method will block until the service is resumed.)
- The message delivery state of any messages already received by the broker is maintained. (For example, transactions are not disrupted and message delivery will resume when the service is resumed.)

The `admin` connection service can never be paused; to pause and resume any other service, use the subcommands `imqcmd pause svc` and `imqcmd resume svc`. The syntax of the `imqcmd pause svc` subcommand is as follows:

```
imqcmd pause svc -n serviceName  
                  [-b hostName:portNumber]
```

For example, the following command pauses the `httpjms` service running on the default broker (host `localhost` at port `7676`):

```
imqcmd pause svc -n httpjms -u admin
```

The `imqcmd resume svc` subcommand resumes operation of a connection service following a pause:

```
imqcmd resume svc -n serviceName  
                  [-b hostName:portNumber]
```

Updating Connection Service Properties

You can use the `imqcmd update svc` subcommand to change the value of one or more of the service properties listed in [Table 6–2](#). See “[Connection Properties](#)” on [page 299](#) for a description of these properties.

TABLE 6–2 Connection Service Properties Updated by Command Utility

Property	Description
port	Port assigned to the service to be updated (does not apply to <code>httpjms</code> or <code>httpsjms</code>) A value of <code>0</code> means the port is dynamically allocated by the Port Mapper.
minThreads	Minimum number of threads assigned to the service
maxThreads	Maximum number of threads assigned to the service

The `imqcmd update svc` subcommand has the following syntax:

```
imqcmd update svc -n serviceName  
                  [-b hostName:portNumber]  
                  [-o property1=value1 [[-o property2=value2]...]
```

For example, the following command changes the minimum number of threads assigned to the `jms` connection service on the default broker (host `localhost` at port `7676`) to `20`:

```
imqcmd update svc -o minThreads=20 -u admin
```

Viewing Connection Service Information

To list the connection services available on a broker, use the `imqcmd list svc` subcommand:

```
imqcmd list svc [-b hostName:portNumber]
```

For example, the following command lists all services on the default broker (host `localhost` at port 7676):

```
imqcmd list svc -u admin
```

[Example 6–1](#) shows an example of the resulting output.

EXAMPLE 6–1 Connection Services Listing

Service Name	Port Number	Service State
admin	41844 (dynamic)	RUNNING
httpjms	-	UNKNOWN
httpsjms	-	UNKNOWN
jms	41843 (dynamic)	RUNNING
ssladmin	dynamic	UNKNOWN
ssljms	dynamic	UNKNOWN

The `imqcmd query svc` subcommand displays information about a single connection service:

```
imqcmd query svc -n serviceName
                  [-b hostName:portNumber]
```

For example, the following command displays information about the `jms` connection service on the default broker (host `localhost` at port 7676):

```
imqcmd query svc -n jms -u admin
```

Example 6–2 shows an example of the resulting output.

EXAMPLE 6–2 Connection Service Information Listing

Service Name	jms
Service State	RUNNING
Port Number	60920 (dynamic)
Current Number of Allocated Threads	0
Current Number of Connections	0
Min Number of Threads	10
Max Number of Threads	1000

To display metrics information about a connection service, use the `imqcmd metrics svc` subcommand:

```
imqcmd metrics svc -n serviceName
                  [-b hostName:portNumber]
                  [-m metricType]
                  [-int interval]
                  [-msp numSamples]
```

The `-m` option specifies the type of metric information to display:

- `ttl` (*default*): Messages and packets flowing into and out of the broker by way of the specified connection service
- `rts`: Rate of flow of messages and packets into and out of the broker per second by way of the specified connection service
- `cxn`: Connections, virtual memory heap, and threads

The `-int` and `-msp` options specify, respectively, the interval (in seconds) at which to display the metrics and the number of samples to display in the output. The default values are 5 seconds and an unlimited number of samples.

For example, the following command displays cumulative totals for messages and packets handled by the default broker (host `localhost` at port `7676`) by way of the `jms` connection service:

```
imqcmd metrics svc -n.jms -m.ttl -u.admin
```

Example 6-3 shows an example of the resulting output.

EXAMPLE 6-3 Connection Service Metrics Listing

Msgs		Msg Bytes		Pkts		Pkt Bytes	
In	Out	In	Out	In	Out	In	Out
164	100	120704	73600	282	383	135967	102127
657	100	483552	73600	775	876	498815	149948

For a more detailed description of the use of the Command utility to report connection service metrics, see [“Connection Service Metrics” on page 358](#).

Managing Connections

The Command utility’s `list cxn` and `query cxn` subcommands display information about individual connections. The subcommand `imqcmd list cxn` lists all connections for a specified connection service:

```
imqcmd list cxn [-svn serviceName]  
                [-b hostName:portNumber]
```

If no service name is specified, all connections are listed. For example, the following command lists all connections on the default broker (host `localhost` at port `7676`):

```
imqcmd list cxn -u admin
```

[Example 6–4](#) shows an example of the resulting output.

EXAMPLE 6–4 Broker Connections Listing

```
Listing all the connections on the broker specified by:
-----
Host                Primary Port
-----
localhost           7676

-----
Connection ID      User      Service  Producers  Consumers  Host
-----
1964412264455443200  guest    jms       0           1         127.0.0.1
1964412264493829311  admin    admin     1           1         127.0.0.1

Successfully listed connections.
```

To display detailed information about a single connection, obtain the connection identifier from `imqcmd list cxn` and pass it to the `imqcmd query cxn` subcommand:

```
imqcmd query cxn  -n connectionID
                  [-b hostName:portNumber]
```

For example, the command

```
imqcmd query cxn  -n 421085509902214374  -u admin
```

produces output like that shown in [Example 6–5](#).

EXAMPLE 6–5 Connection Information Listing

Connection ID	421085509902214374
User	guest
Service	jms
Producers	0
Consumers	1
Host	111.22.333.444
Port	60953
Client ID	
Client Platform	

The `imqcmd destroy cxn` subcommand destroys a connection:

```
imqcmd destroy cxn -n connectionID  
                    [-b hostName:portNumber]
```

For example, the command

```
imqcmd destroy cxn -n 421085509902214374 -u admin
```

destroys the connection shown in [Example 6–5](#).

Managing Message Delivery

A Message Queue message is routed to its consumer clients by way of a *physical destination* on a message broker. The broker manages the memory and persistent storage associated with the physical destination and configures its behavior. The broker also manages memory at a system-wide level, to assure that sufficient resources are available to support all destinations.

Message delivery also involves the maintenance of state information needed by the broker to route messages to consumers and to track acknowledgements and transactions.

This chapter provides information needed to manage message delivery, and includes the following topics:

- [“Configuring and Managing Physical Destinations” on page 105](#)
- [“Managing Broker System-Wide Memory” on page 119](#)
- [“Managing Durable Subscriptions” on page 121](#)
- [“Managing Transactions” on page 122](#)

Configuring and Managing Physical Destinations

This section describes how to use the Message Queue Command utility (`imqcmd`) to manage physical destinations. It includes discussion of a specialized physical destination managed by the broker, the *dead message queue*, whose properties differ somewhat from those of other destinations.

Note – In a broker cluster, you create a physical destination on one broker and the cluster propagates it to all the others. Because the brokers cooperate to route messages across the cluster, client applications can consume messages from destinations on any broker in the cluster. However the persistence and acknowledgment of a message is managed only by the broker to which a message was originally produced.

This section covers the following topics regarding the management of physical destinations:

- “Command Utility Subcommands for Physical Destination Management” on page 106
- “Creating and Destroying Physical Destinations” on page 107
- “Pausing and Resuming a Physical Destination” on page 110
- “Purging a Physical Destination” on page 111
- “Updating Physical Destination Properties” on page 112
- “Viewing Physical Destination Information” on page 112
- “Managing Physical Destination Disk Utilization” on page 116
- “Using the Dead Message Queue” on page 118

Note – For provider independence and portability, client applications typically use *destination administered objects* to interact with physical destinations. [Chapter 11, “Managing Administered Objects,”](#) describes how to configure such administered objects for use by client applications. For a general conceptual introduction to physical destinations, see the *Message Queue Technical Overview*.

Command Utility Subcommands for Physical Destination Management

The Message Queue Command utility (`imqcmd`) enables you to manage physical destinations interactively from the command line. See [Chapter 15, “Command Line Reference,”](#) for general reference information about the syntax, subcommands, and options of the `imqcmd` command, and [Chapter 17, “Physical Destination Property Reference,”](#) for specific information on the configuration properties used to specify physical destination behavior.

[Table 7–1](#) lists the `imqcmd` subcommands for physical destination management. For full reference information about these subcommands, see [Table 15–7](#).

TABLE 7–1 Physical Destination Subcommands for the Command Utility

Subcommand	Description
<code>create dst</code>	Create physical destination
<code>destroy dst</code>	Destroy physical destination
<code>pause dst</code>	Pause message delivery for physical destination
<code>resume dst</code>	Resume message delivery for physical destination
<code>purge dst</code>	Purge all messages from physical destination
<code>compact dst</code>	Compact physical destination
<code>update dst</code>	Set physical destination properties

TABLE 7-1 Physical Destination Subcommands for the Command Utility (Continued)

Subcommand	Description
<code>list dst</code>	List physical destinations
<code>query dst</code>	List physical destination property values
<code>metrics dst</code>	Display physical destination metrics

Creating and Destroying Physical Destinations

The subcommand `imqcmd create dst` creates a new physical destination:

```
imqcmd create dst -t destType -n destName
[ [-o property=value] ... ]
```

You supply the destination type (q for a queue or t for a topic) and the name of the destination.

Naming Destinations

Destination names must conform to the rules described below for queue and topic destinations.

Supported Queue Destination Names

Queue destination names must conform to the following rules:

- It must contain only alphanumeric characters.
- It must not contain spaces.
- It must begin with an alphabetic character (A–Z, a–z), an underscore (`_`), or a dollar sign (`$`).
- It must not begin with the characters `mq`.

For example, the following command creates a queue destination named `XQueue`:

```
imqcmd create dst -t q -n XQueue
```

Supported Topic Destination Names

Topic destination names must conform to the same rules as queue destinations, as specified in [“Supported Queue Destination Names” on page 107](#), except that Message Queue also supports, in addition, topic destination names that include wildcard characters, representing multiple destinations. These symbolic names allow publishers to publish messages to multiple topics and subscribers to consume messages from multiple topics. Using symbolic names, you can create destinations, as needed, consistent with the wildcard naming scheme. Publishers and subscribers automatically publish to and consume from any added destinations that match the symbolic names. (Wildcard topic subscribers are more common than publishers.)

The format of a symbolic topic destination name consists of multiple segments, in which wildcard characters (*, **, >) can represent one or more segments of the name. For example, suppose you have a topic destination naming scheme as follows:

size.color.shape

where the topic name segments can have the following values:

- *size*: large, medium, small, ...
- *color*: red, green, blue, ...
- *shape*: circle, triangle, square, ...

Message Queue supports the following wildcard characters:

- * matches a single segment
- ** matches one or more segments
- > matches any number of successive segments

You can therefore indicate multiple topic destinations as follows:

large.*.circle would represent:

```
large.red.circle
large.green.circle
...
```

**.square would represent all names ending in .square, for example:

```
small.green.square
medium.blue.square
...
```

small.> would represent all destination names starting with small., for example:

```
small.blue.circle
small.red.square
...
```

To use this multiple destination feature, you create topic destinations using a naming scheme similar to that described above. For example, the following command creates a topic destination named `large.green.circle`:

```
imqcmd create dst -t t -n large.green.circle
```

Client applications can then create wildcard publishers or wildcard consumers using symbolic destination names, as shown in the following examples:

EXAMPLE 7-1 Wildcard Publisher

```
...
String DEST_LOOKUP_NAME = "large.*.circle";
Topic t = (Destination) ctx.lookup(DEST_LOOKUP_NAME);
TopicPublisher myPublisher = mySession.createPublisher(t);
myPublisher.send(myMessage);
```

In this example, the broker will place a copy of the message in any destination that matches the symbolic name `large.*.circle`

EXAMPLE 7-2 Wildcard Subscriber

```
...
String DEST_LOOKUP_NAME = "**.square";
Topic t = (Destination) ctx.lookup(DEST_LOOKUP_NAME);
TopicSubscriber mySubscriber = mySession.createSubscriber(t);
Message m = mySubscriber.receive();
```

In this example, a subscriber will be created if there is at least one destination that matches the symbolic name `**.*.square` and will receive messages from all destinations that match that symbolic name. If there are no destinations matching the symbolic name, the subscriber will not be registered with the broker until such a destination exists.

If you create additional destinations that match a symbolic name, then wildcard publishers created using that symbolic name will subsequently publish to that destination and wildcard subscribers created using that symbolic name will subsequently receive messages from that destination.

In addition, Message Queue administration tools, in addition to reporting the total number of publishers (producers) and subscribers (consumers) for a topic destination, will also report the number of publishers that are wildcard publishers (including their corresponding symbolic destination names) and the number of subscribers that are wildcard subscribers (including their symbolic destination names), if any. See [“Viewing Physical Destination Information” on page 112](#).

Setting Property Values

The `imqcmd create dst` command may also optionally include any property values you wish to set for the destination, specified with the `-o` option. For example, the following command creates a topic destination named `hotTopic` with a maximum message length of 5000 bytes:

```
imqcmd create dst -t t -n hotTopic -o maxBytesPerMsg=5000
```

See [Chapter 17, “Physical Destination Property Reference,”](#) for reference information about the physical destination properties that can be set with this option. (For auto-created destinations, you set default property values in the broker’s instance configuration file; see [Table 16–3](#) for information on these properties.)

Destroying Destinations

To destroy a physical destination, use the `imqcmd destroy dst` subcommand:

```
imqcmd destroy dest -t destType -n destName
```

This purges all messages at the specified destination and removes it from the broker; the operation is not reversible.

For example, the following command destroys the queue destination named `curlyQueue`:

```
imqcmd destroy dest -t q -n curlyQueue -u admin
```

Note – You cannot destroy the dead message queue.

Pausing and Resuming a Physical Destination

Pausing a physical destination temporarily suspends the delivery of messages from producers to the destination, from the destination to consumers, or both. This can be useful, for instance, to prevent destinations from being overwhelmed when messages are being produced much faster than they are consumed. You must also pause a physical destination before compacting it (see [“Managing Physical Destination Disk Utilization” on page 116](#)).

To pause the delivery of messages to or from a physical destination, use the `imqcmd pause dst` subcommand:

```
imqcmd pause dest [-t destType -n destName]  
                  [-pst pauseType]
```

If you omit the destination type and name (`-t` and `-n` options), all physical destinations will be paused. The pause type (`-pst`) specifies what type of message delivery to pause:

PRODUCERS	Pause delivery from message producers to the destination
CONSUMERS	Pause delivery from the destination to message consumers
ALL	Pause all message delivery (both producers and consumers)

If no pause type is specified, all message delivery will be paused.

For example, the following command pauses delivery from message producers to the queue destination `curlyQueue`:

```
imqcmd pause dst -t q -n curlyQueue -pst PRODUCERS -u admin
```

The following command pauses delivery to message consumers from the topic destination `hotTopic`:

```
imqcmd pause dst -t t -n hotTopic -pst CONSUMERS -u admin
```

This command pauses all message delivery to and from all physical destinations:

```
imqcmd pause dst -u admin
```

Note – In a broker cluster, since each broker in the cluster has its own instance of each physical destination, you must pause each such instance individually.

The `imqcmd resume dst` subcommand resumes delivery to a paused destination:

```
imqcmd resume dest [-t destType -n destName]
```

For example, the following command resumes message delivery to the queue destination `curlyQueue`:

```
imqcmd resume dst -t q -n curlyQueue -u admin
```

If no destination type and name are specified, all destinations are resumed. This command resumes delivery to all physical destinations:

```
imqcmd resume dst -u admin
```

Purging a Physical Destination

Purging a physical destination deletes all messages it is currently holding. You might want to do this when a destination's accumulated messages are taking up too much of the system's resources, such as when a queue is receiving messages but has no registered consumers to which to deliver them, or when a topic's durable subscribers remain inactive for long periods of time.

To purge a physical destination, use the `imqcmd purge dst` subcommand:

```
imqcmd purge dst -t destType -n destName
```

For example, the following command purges all accumulated messages from the topic destination `hotTopic`:

```
imqcmd purge dst -t t -n hotTopic -u admin
```

Note – In a broker cluster, since each broker in the cluster has its own instance of each physical destination, you must purge each such instance individually.

Tip – When restarting a broker that has been shut down, you can use the Broker utility's `-reset messages` option to clear out its stale messages: for example,

```
imqbrokerd -reset messages -u admin
```

This saves you the trouble of purging physical destinations after restarting the broker.

Updating Physical Destination Properties

The subcommand `imqcmd update dst` changes the values of specified properties of a physical destination:

```
imqcmd update dst -t destType -n destName
                  -o property1=value1 [ [-o property2=value2] ... ]
```

The properties to be updated can include any of those listed in [Table 17-1](#) (with the exception of the `isLocalOnly` property, which cannot be changed once the destination has been created). For example, the following command changes the `maxBytesPerMsg` property of the queue destination `curlyQueue` to `1000` and the `maxNumMsgs` property to `2000`:

```
imqcmd update dst -t q -n curlyQueue -u admin
                  -o maxBytesPerMsg=1000
                  -o maxNumMsgs=2000
```

Note – The *type* of a physical destination is not an updatable property; you cannot use the `imqcmd update dst` subcommand to change a queue to a topic or a topic to a queue.

Viewing Physical Destination Information

To list the physical destinations on a broker, use the `imqcmd list dst` subcommand:

```
imqcmd list dst -b hostName:portNumber [-t destType] [-tmp]
```


This lists all physical destinations on the broker identified by *hostName* and *portNumber* of the type (queue or topic) specified by *destType*. If the `-t` option is omitted, both queues and topics are listed. For example, the following command lists all physical destinations on the broker running on host *myHost* at port number 4545:

```
imqcmd list dst -b myHost:4545
```

Note – The list of queue destinations always includes the dead message queue (`mq.sys.dmq`) in addition to any other queue destinations currently existing on the broker.

If you specify the `-tmp` option, temporary destinations are listed as well. These are destinations created by clients, normally for the purpose of receiving replies to messages sent to other clients.

The `imqcmd query dst` subcommand displays information about a single physical destination:

```
imq query dst -t destType -n destName
```

For example, the following command displays information about the queue destination `curlyQueue`:

```
imqcmd query dst -t q -n curlyQueue -u admin
```

Example 7-3 shows an example of the resulting output. You can use the `imqcmd update dst` subcommand (see “Updating Physical Destination Properties” on page 112) to change the value of any of the properties listed.

EXAMPLE 7-3 Physical Destination Information Listing

```
-----
Destination Name    Destination Type
-----
large.green.circle  Topic

On the broker specified by:

-----
Host                Primary Port
-----
localhost           7676

Destination Name    large.green.circle
Destination Type     Topic
Destination State    RUNNING
Created Administratively true

Current Number of Messages
  Actual              0
  Remote              0
  Held in Transaction 0
Current Message Bytes
  Actual              0
  Remote              0
  Held in Transaction 0
Current Number of Producers      0
Current Number of Producer Wildcards 0
Current Number of Consumers      1
Current Number of Consumer Wildcards 1
  large.*.circle (1)

Max Number of Messages    unlimited (-1)
Max Total Message Bytes   unlimited (-1)
Max Bytes per Message     unlimited (-1)
Max Number of Producers   100

Limit Behavior            REJECT_NEWEST
Consumer Flow Limit       1000
Is Local Destination      false
Use Dead Message Queue    true
XML schema validation enabled false
XML schema URI List       -
Reload XML schema on failure false
```

For destinations in a broker cluster, it is often helpful to know how many messages in a destination are local (produced to the local broker) and how many are remote (produced to a remote broker). Hence, `imqcmd query dst` reports, in addition to the number and total message bytes of messages in the destination, the number and total bytes of messages that are sent to the destination from remote brokers in the cluster.

For topic destinations, `imqcmd query dst` reports the number of publishers that are wildcard publishers (including their corresponding symbolic destination names) and the number of subscribers that are wildcard subscribers (including their symbolic destination names), if any.

To display metrics information about a physical destination, use the `imqcmd metrics dst` subcommand:

```
imqcmd metrics dst  -t destType -n destName
                    [-m metricType]
                    [-int interval]
                    [-msp numSamples]
```

The `-m` option specifies the type of metric information to display:

- `tll` (*default*): Messages and packets flowing into and out of the destination and residing in memory
- `rts`: Rate of flow of messages and packets into and out of the destination per second, along with other rate information
- `con`: Metrics related to message consumers
- `dsk`: Disk usage

The `-int` and `-msp` options specify, respectively, the interval (in seconds) at which to display the metrics and the number of samples to display in the output. The default values are 5 seconds and an unlimited number of samples.

For example, the following command displays cumulative totals for messages and packets handled by the queue destination `curlyQueue`:

```
imqcmd metrics dst  -t q  -n curlyQueue  -m tll  -u admin
```

[Example 7–4](#) shows an example of the resulting output.

EXAMPLE 7–4 Physical Destination Metrics Listing

Msgs		Msg Bytes		Msg Count			Total Msg Bytes (k)			Largest
In	Out	In	Out	Current	Peak	Avg	Current	Peak	Avg	Msg (k)
3128	3066	1170102	1122340	128	409	29	46	145	10	< 1
4858	4225	1863159	1635458	144	201	33	53	181	42	< 1
2057	1763	820804	747200	84	377	16	71	122	79	< 1

For a more detailed description of the use of the Command utility to report physical destination metrics, see [“Physical Destination Metrics” on page 359](#).

Managing Physical Destination Disk Utilization

Because of the way message storage is structured in a file-based persistent data store (see [“File-Based Persistence Properties” on page 126](#)), disk space can become fragmented over time, resulting in inefficient utilization of the available resources. Message Queue’s Command utility (`imqcmd`) provides subcommands for monitoring disk utilization by physical destinations and for reclaiming unused disk space when utilization drops.

To monitor a physical destination’s disk utilization, use the `imqcmd metrics dst` subcommand:

```
imqcmd metrics dst -m dsk -t destType -n destName
```

This displays the total number of bytes of disk space reserved for the destination’s use, the number of bytes currently in use to hold active messages, and the percentage of available space in use (the *disk utilization ratio*). For example, the following command displays disk utilization information for the queue destination `curlyQueue`:

```
imqcmd metrics dst -m dsk -t q -n curlyQueue -u admin
```

[Example 7–5](#) shows an example of the resulting output.

EXAMPLE 7–5 Destination Disk Utilization Listing

Reserved	Used	Utilization Ratio
<hr/>		
804096	675533	84
1793024	1636222	91
2544640	2243808	88

The disk utilization pattern depends on the characteristics of the messaging application using a particular physical destination. Depending on the flow of messages into and out of the destination and their relative size, the amount of disk space reserved might grow over time. If messages are produced at a higher rate than they are consumed, free records should generally be reused and the utilization ratio should be on the high side. By contrast, if the rate of message production is comparable to or lower than the consumption rate, the utilization ratio will likely be low.

As a rule, you want the reserved disk space to stabilize and the utilization ratio to remain high. If the system reaches a steady state in which the amount of reserved disk space remains more or less constant with utilization above 75%, there is generally no need to reclaim unused disk space. If the reserved space stabilizes at a utilization rate below 50%, you can use the `imqcmd compact dst` subcommand to reclaim the disk space occupied by free records:

```
compact dst [-t destType -n destName]
```

This compacts the file-based data store for the designated physical destination. If no destination type and name are specified, all physical destinations are compacted.

You must pause a destination (with the `imqcmd pause` subcommand) before compacting it, and resume it (with `imqcmd resume`) afterward (see [“Pausing and Resuming a Physical Destination” on page 110](#)):

```
imqcmd pause dst -t q -n curlyQueue -u admin
imqcmd compact dst -t q -n curlyQueue -u admin
imqcmd resume dst -t q -n curlyQueue -u admin
```

Tip – If a destination’s reserved disk space continues to increase over time, try reconfiguring its `maxNumMsgs`, `maxBytesPerMsg`, `maxTotalMsgBytes`, and `limitBehavior` properties (see [“Physical Destination Properties” on page 333](#)).

Using the Dead Message Queue

The *dead message queue*, `mq.sys.dmq`, is a system-created physical destination that holds the dead messages of a broker's physical destinations. The dead message queue is a tool for monitoring, tuning system efficiency, and troubleshooting. For a definition of the term *dead message* and a more detailed introduction to the dead message queue, see the *Message Queue Technical Overview*.

The broker automatically creates a dead message queue when it starts. The broker places messages on the queue if it cannot process them or if their time-to-live has expired. In addition, other physical destinations can use the dead message queue to hold discarded messages. This can provide information that is useful for troubleshooting the system.

Managing the Dead Message Queue

The physical destination configuration property `useDMQ` controls a destination's use of the dead message queue. Physical destinations are configured to use the dead message queue by default; to disable a destination from using it, set the destination's `useDMQ` property to `false`:

```
imqcmd update dst -t q -n curlyQueue -o useDMQ=false
```

You can enable or disable the use of the dead message queue for all auto-created physical destinations on a broker by setting the broker's `imq.autocreate.destination.useDMQ` broker property:

```
imqcmd update bkr -o imq.autocreate.destination.useDMQ=false
```

You can manage the dead message queue with the Message Queue Command utility (`imqcmd`) just as you manage other queues, but with some differences. For example, because the dead message queue is system-created, you cannot create, pause, or destroy it. Also, as shown in [Table 7-2](#), default values for the dead message queue's configuration properties sometimes differ from those of ordinary queues.

TABLE 7-2 Dead Message Queue Treatment of Physical Destination Properties

Property	Variant Treatment by Dead Message Queue
<code>maxNumMsgs</code>	Default value is <code>1000</code> , rather than <code>-1</code> (unlimited) as for ordinary queues.
<code>maxTotalMsgBytes</code>	Default value is <code>10m</code> (10 megabytes), rather than <code>-1</code> (unlimited) as for ordinary queues.
<code>limitBehavior</code>	Default value is <code>REMOVE_OLDEST</code> , rather than <code>REJECT_NEWEST</code> as for ordinary queues. <code>FLOW_CONTROL</code> is not supported for the dead message queue.

TABLE 7-2 Dead Message Queue Treatment of Physical Destination Properties (Continued)

Property	Variant Treatment by Dead Message Queue
maxNumProducers	Does not apply to the dead message queue.
isLocalOnly	Permanently set to false in broker clusters; the dead message queue in a cluster is always a global physical destination.
localDeliveryPreferred	Does not apply to the dead message queue.

Tip – By default, the dead message queue stores entire messages. If you do not plan to restore dead messages, you can reduce the size of the dead message queue by setting the broker's `imq.destination.DMQ.truncateBody` property to `true`:

```
imqcmd update bkr -o imq.destination.DMQ.truncateBody=true
```

This will discard the body of all messages and retain only the headers and property data.

Enabling Dead Message Logging

The broker configuration property `logDeadMsgs` controls the logging of events related to the dead message queue. When dead message logging is enabled, the broker will log the following events:

- A message is moved to the dead message queue.
- A message is discarded from the dead message queue (or from any physical destination that does not use the dead message queue).
- A physical destination reaches its limits.

Dead message logging is disabled by default. The following command enables it:

```
imqcmd update bkr -o imq.destination.logDeadMsgs=true
```

Note – Dead message logging applies to all physical destinations that use the dead message queue. You cannot enable or disable logging for an individual physical destination.

Managing Broker System-Wide Memory

Once clients are connected to the broker, the routing and delivery of messages can proceed. In this phase, the broker is responsible for creating and managing different types of physical destinations, ensuring a smooth flow of messages, and using resources efficiently. You can use the broker configuration properties described under [“Routing and Delivery Properties” on page 302](#) to manage these tasks in a way that suits your application’s needs.

The performance and stability of a broker depend on the system resources (such as memory) available and how efficiently they are utilized. You can set configuration properties to prevent the broker from becoming overwhelmed by incoming messages or running out of memory. These properties function at three different levels to keep the message service operating as resources become scarce:

- **Systemwide message limits** apply collectively to all physical destinations on the system. These include the maximum number of messages held by a broker (`imq.system.max_count`) and the maximum total number of bytes occupied by such messages (`imq.system.max_size`). If either of these limits is reached, the broker will reject any new messages until the pending messages fall below the limit. There is also a limit on the maximum size of an individual message (`imq.message.max_size`) and a time interval at which expired messages are reclaimed (`imq.message.expiration.interval`).
- **Individual destination limits** regulate the flow of messages to a specific physical destination. The configuration properties controlling these limits are described in [Chapter 17, “Physical Destination Property Reference.”](#) They include limits on the number and size of messages the destination will hold, the number of message producers and consumers that can be created for it, and the number of messages that can be batched together for delivery to the destination.

The destination can be configured to respond to memory limits by slowing down the delivery of message by message producers, by rejecting new incoming messages, or by throwing out the oldest or lowest-priority existing messages. Messages deleted from the destination in this way may optionally be moved to the dead message queue rather than discarded outright; the broker property `imq.destination.DMQ.truncateBody` controls whether the entire message body is saved in the dead message queue, or only the header and property data.

As a convenience during application development and testing, you can configure a message broker to create new physical destinations automatically whenever a message producer or consumer attempts to access a nonexistent destination. The broker properties summarized in [Table 16–3](#) parallel the ones just described, but apply to such *auto-created destinations* instead of administratively created ones.

- **System memory thresholds** define levels of memory usage at which the broker takes increasingly serious action to prevent memory overload. Four such usage levels are defined:
 - **Green:** Plenty of memory is available.
 - **Yellow:** Broker memory is beginning to run low.
 - **Orange:** The broker is low on memory.
 - **Red:** The broker is out of memory.

The memory utilization percentages defining these levels are specified by the broker properties `imq.green.threshold`, `imq.yellow.threshold`, `imq.orange.threshold`, and `imq.red.threshold`, respectively; the default values are 0% for green, 80% for yellow, 90% for orange, and 98% for red.

As memory usage advances from one level to the next, the broker responds progressively, first by swapping messages out of active memory into persistent storage and then by throttling back producers of nonpersistent messages, eventually stopping the flow of messages into the broker. (Both of these measures degrade broker performance.) The throttling back of message production is done by limiting the size of each batch delivered to the number of messages specified by the properties `imq.resourceState.count`, where *resourceState* is green, yellow, orange, or red, respectively.

The triggering of these system memory thresholds is a sign that systemwide and destination message limits are set too high. Because the memory thresholds cannot always catch potential memory overloads in time, you should not rely on them to control memory usage, but rather reconfigure the system-wide and destination limits to optimize memory resources.

Managing Durable Subscriptions

Message Queue clients subscribing to a topic destination can register as *durable subscribers*. The corresponding durable subscription has a unique, persistent identity and requires the broker to retain messages addressed to it even when its message consumer (the durable subscriber) becomes inactive. Ordinarily, the broker may delete a message held for a durable subscriber only when the message expires.

The Message Queue Command utility provides subcommands for managing a broker's durable subscriptions in the following ways:

- Listing durable subscriptions
- Purging all messages for a durable subscription
- Destroying a durable subscription

To list durable subscriptions for a specified physical destination, use the `imqcmd list dur` subcommand:

```
imqcmd list dur -d topicName
```

For example, the following command lists all durable subscriptions to the topic `SPQuotes` on the default broker (host `localhost` at port `7676`):

```
imqcmd list dur -d SPQuotes
```

The resulting output lists the name of each durable subscription to the topic, the client identifier to which it belongs, its current state (active or inactive), and the number of messages currently queued to it. [Example 7–6](#) shows an example.

EXAMPLE 7–6 Durable Subscription Information Listing

Name	Client ID	Number of Messages	Durable Sub State

myDurable	myClientID	1	INACTIVE

The `imqcmd purge dur` subcommand purges all messages for a specified durable subscriber and client identifier:

```
imqcmd purge dur -n subscriberName
                  -c clientID
```

For example, the following command purges all messages for the durable subscription listed in [Example 7–6](#):

```
imqcmd purge dur -n myCurable -c myClientID
```

The `imqcmd destroy dur` subcommand destroys a durable subscription, specified by its subscriber name and client identifier:

```
imqcmd destroy dur -n subscriberName
                   -c clientID
```

For example, the following command destroys the durable subscription listed in [Example 7–6](#):

```
imqcmd destroy dur -n myCurable -c myClientID
```

Managing Transactions

All transactions initiated by client applications are tracked by the broker. These can be local Message Queue transactions or distributed transactions managed by a distributed transaction manager.

Each transaction is identified by a unique 64-bit Message Queue *transaction identifier*. Distributed transactions also have a *distributed transaction identifier (XID)*, up to 128 bytes long, assigned by the distributed transaction manager. Message Queue maintains the association between its own transaction identifiers and the corresponding XIDs.

The `imqcmd list txn` subcommand lists the transactions being tracked by a broker:

```
imqcmd list txn
```

This lists all transactions on the broker, both local and distributed. For each transaction, it shows the transaction ID, state, user name, number of messages and acknowledgments, and creation time. [Example 7-7](#) shows an example of the resulting output.

EXAMPLE 7-7 Broker Transactions Listing

Transaction ID	State	User name	# Msgs/ # Acks	Creation time
64248349708800	PREPARED	guest	4/0	1/30/02 10:08:31 AM
64248371287808	PREPARED	guest	0/4	1/30/02 10:09:55 AM

To display detailed information about a single transaction, obtain the transaction identifier from `imqcmd list txn` and pass it to the `imqcmd query txn` subcommand:

```
imqcmd query txn -n transactionID
```

This displays the same information as `imqcmd list txn`, along with the client identifier, connection identification, and distributed transaction identifier (XID). For example, the command

```
imqcmd query txn -n 64248349708800
```

produces output like that shown in [Example 7-8](#).

EXAMPLE 7-8 Transaction Information Listing

Client ID	
Connection	guest@192.18.116.219:62209->jms:62195
Creation time	1/30/02 10:08:31 AM
Number of acknowledgments	0
Number of messages	4
State	PREPARED
Transaction ID	64248349708800
User name	guest
XID	6469706F6C7369646577696E6465723130313234313431313030373230

If a broker fails, it is possible that a distributed transaction could be left in the PREPARED state without ever having been committed. Until such a transaction is committed, its messages will not be delivered and its acknowledgments will not be processed. Hence, as an administrator,

you might need to monitor such transactions and commit them or roll them back manually. For example, if the broker's `imq.transaction.autorollback` property (see [Table 16–2](#)) is set to `false`, you must manually commit or roll back non-distributed transactions and unrecoverable distributed transactions found in the `PREPARED` state at broker startup, using the Command utility's `commit txn` or `rollback txn` subcommand:

```
imqcmd commit txn -n transactionID
```

```
imqcmd rollback txn -n transactionID
```

For example, the following command commits the transaction listed in [Example 7–8](#):

```
imqcmd commit txn -n64248349708800
```

Note – Only transactions in the `PREPARED` state can be committed. However, transaction in the `STARTED`, `FAILED`, `INCOMPLETE`, `COMPLETE`, and `PREPARED` states can be rolled back. You should do so only if you know that the transaction has been left in this state by a failure and is not in the process of being committed by the distributed transaction manager.

Configuring Persistence Services

For a broker to recover in case of failure, it needs to re-create the state of its message delivery operations. To do this, the broker must save state information to a *persistent data store*. When the broker restarts, it uses the saved data to re-create destinations and durable subscriptions, recover persistent messages, roll back open transactions, and rebuild its routing table for undelivered messages. It can then resume message delivery.

A persistent data store is thus a key aspect of providing for reliable message delivery. This chapter describes the two different persistence implementations supported by the Message Queue broker and how to set each of them up:

- [“Introduction to Persistence Services” on page 125](#)
- [“File-Based Persistence” on page 126](#)
- [“JDBC-Based Persistence” on page 128](#)
- [“Data Store Formats” on page 132](#)

Introduction to Persistence Services

A broker’s persistent data store holds information about physical destinations, durable subscriptions, messages, transactions, and acknowledgments.

Message Queue supports both file-based and JDBC-based persistence modules, as shown in the following figure. File-based persistence uses individual files to store persistent data; JDBC-based persistence uses the Java Database Connectivity (JDBC) interface to connect the broker to a JDBC-based data store. While file-based persistence is generally faster than JDBC-based persistence, some users prefer the redundancy and administrative control provided by a JDBC database. The broker configuration property `imq.persist.store` (see [Table 16–4](#)) specifies which of the two persistence modules (`file` or `jdbc`) to use.

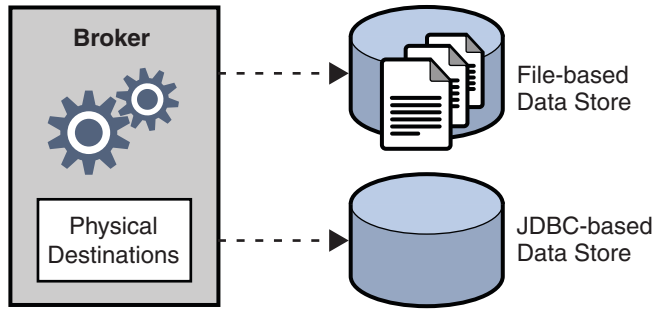


FIGURE 8-1 Persistent Data Stores

Message Queue brokers are configured by default to use a file-based persistent store, but you can reconfigure them to plug in any data store accessible through a JDBC-compliant driver. The broker configuration property `imq.persist.store` (see [Table 16-4](#)) specifies which of the two forms of persistence to use.

File-Based Persistence

By default, Message Queue uses a file-based data store, in which individual files store persistent data (such as messages, destinations, durable subscriptions, transactions, and routing information).

The file-based data store is located in a directory identified by the name of the broker instance (*instanceName*) to which the data store belongs:

```
.../instances/instanceName/fs370
```

(See [Appendix A, “Platform-Specific Locations of Message Queue Data,”](#) for the location of the `instances` directory.) Each destination on the broker has its own subdirectory holding messages delivered to that destination.

Note – Because the data store can contain messages of a sensitive or proprietary nature, you should secure the `.../instances/instanceName/fs370` directory against unauthorized access; see [“Securing a File-Based Data Store” on page 128](#).

File-Based Persistence Properties

Broker configuration properties related to file-based persistence are listed under [“File-Based Persistence Properties” on page 307](#). These properties let you configure various aspects of how the file-based data store behaves.

All persistent data other than messages is stored in separate files: one file each for destinations, durable subscriptions, and transaction state information. Most messages are stored in a single file consisting of variable-size records. You can compact this file to alleviate fragmentation as messages are added and removed (see [“Managing Physical Destination Disk Utilization” on page 116](#)). In addition, messages above a certain threshold size are stored in their own individual files rather than in the variable-sized record file. You can configure this threshold size with the broker property `imq.persist.file.message.max_record_size`.

The broker maintains a file pool for these individual message files: instead of being deleted when it is no longer needed, a file is returned to the pool of free files in its destination directory so that it can later be reused for another message. The broker property `imq.persist.file.destination.message.filepool.limit` specifies the maximum number of files in the pool. When the number of individual message files for a destination exceeds this limit, files will be deleted when no longer needed instead of being returned to the pool.

When returning a file to the file pool, the broker can save time at the expense of storage space by simply tagging the file as available for reuse without deleting its previous contents. You can use the `imq.persist.file.message.filepool.cleanratio` broker property to specify the percentage of files in each destination’s file pool that should be maintained in a “clean” (empty) state rather than simply marked for reuse. The higher you set this value, the less space will be required for the file pool, but the more overhead will be needed to empty the contents of files when they are returned to the pool. If the broker’s `imq.persist.file.message.cleanup` property is true, all files in the pool will be emptied at broker shutdown, leaving them in a clean state; this conserves storage space but slows down the shutdown process.

In writing data to the data store, the operating system has some leeway in whether to write the data synchronously or “lazily” (asynchronously). Lazy storage can lead to data loss in the event of a system crash, if the broker believes the data to have been written to the data store when it has not. To ensure absolute reliability (at the expense of performance), you can require that all data be written synchronously by setting the broker property `imq.persist.file.sync.enabled` to true. In this case, the data is guaranteed to be available when the system comes back up after a crash, and the broker can reliably resume operation.

Configuring a File-Based Data Store

A file-based data store is automatically created when you create a broker instance. However, you can configure the data store using the properties described in [“File-Based Persistence Properties” on page 126](#).

For example, by default, Message Queue performs asynchronous write operations to disk. However, to attain the highest reliability, you can set the broker property `imq.persist.file.sync` to write data synchronously instead. See [Table 16–5](#).

When you start a broker instance, you can use the `imqbrokerd` command’s `--reset` option to clear the file-based data store. For more information about this option and its suboptions, see [“Broker Utility” on page 278](#).

Securing a File-Based Data Store

The persistent data store can contain, among other information, message files that are being temporarily stored. Since these messages may contain proprietary information, it is important to secure the data store against unauthorized access. This section describes how to secure data in a file-based data store.

A broker using file-based persistence writes persistent data to a flat-file data store whose location is platform-dependent (see [Appendix A, “Platform-Specific Locations of Message Queue Data”](#)):

```
.../instances/instanceName/fs370
```

where *instanceName* is a name identifying the broker instance. This directory is created when the broker instance is started for the first time. The procedure for securing this directory depends on the operating system platform on which the broker is running:

- On Solaris and Linux, the directory's permissions are determined by the file mode creation mask (*umask*) of the user who started the broker instance. Hence, permission to start a broker instance and to read its persistent files can be restricted by setting the mask appropriately. Alternatively, an administrator (superuser) can secure persistent data by setting the permissions on the *instances* directory to *700*.
- On Windows, the directory's permissions can be set using the mechanisms provided by the Windows operating system. This generally involves opening a Properties dialog for the directory.

JDBC-Based Persistence

Instead of using a file-based data store, you can set up a broker to access any data store accessible through a JDBC-compliant driver. This involves setting the appropriate JDBC-related broker configuration properties and using the Database Manager utility (*imqdbmgr*) to create the proper database schema. See [“Configuring a JDBC-Based Data Store” on page 130](#) for specifics.

JDBC-Based Persistence Properties

The full set of properties for configuring a broker to use a JDBC database are listed in [Table 16–6](#). You can specify these properties either in the instance configuration file (*config.properties*) of each broker instance or by using the *-D* command line option to the Broker utility (*imqbrokerd*) or the Database Manager utility (*imqdbmgr*).

In practice, however, JDBC properties are preconfigured by default, depending on the database vendor being used for the data store. The property values are set in the *default.properties* file, and only need to be explicitly set if you are overriding the default values. In general, you only need to set the following properties:

- `imq.persist.store`
This property specifies that a JDBC-based data store (as opposed to the default file-based data store) is used to store persistent data.
- `imq.persist.jdbc.dbVendor`
This property identifies the database vendor being used for the data store; all of the remaining properties are qualified by this vendor name.
- `imq.persist.jdbcvendorName.user`
This property specifies the user name to be used by the broker in accessing the database.
- `imq.persist.jdbcvendorName.password`
This property specifies the password for accessing the database, if required; `imq.persist.jdbc.vendorName.needpassword` is a boolean flag specifying whether a password is needed. For security reasons, the database access password should be specified only in a password file referenced with the `-passfile` command line option; if no such password file is specified, the `imqbrokerd` and `imqdbmgr` commands will prompt for the password interactively.
- `imq.persist.jdbc.vendorName.property.propName`
This set of properties represents any additional, vendor-specific properties that are required.
- `imq.persist.jdbc.vendorName.tableoption`
Specifies the vendor-specific options passed to the database when creating the table schema.

EXAMPLE 8-1 Broker Properties for MySQL Database

```
imq.persist.store=jdbc
imq.persist.jdbc.dbVendor=mysql
imq.persist.jdbc.mysql.user=userName
imq.persist.jdbc.mysql.password=password
imq.persist.jdbc.mysql.property.url=jdbc:mysql://hostName:port/dataBase
```

If you expect to have messages that are larger than 1 MB, configure MySQL's `max_allowed_packet` variable accordingly when starting the database. For more information see Appendix B of the *MySQL 5.0 Reference Manual*.

EXAMPLE 8-2 Broker Properties for HADB Database

```
imq.persist.store=jdbc
imq.persist.jdbc.dbVendor=hadb
imq.persist.jdbc.hadb.user=userName
imq.persist.jdbc.hadb.password=password
imq.persist.jdbc.hadb.property.serverlist=hostName:port,hostName:port,...
```

You can obtain the server list using the `hadbm get jdbcURL` command.

In addition, in a high-availability broker cluster, in which a JDBC database is shared by multiple broker instances, each broker must be uniquely identified in the database (unnecessary for an embedded database, which stores data for only one broker instance). The configuration property `imq.brokerid` specifies a unique instance identifier to be appended to the names of database tables for each broker. See [“High-Availability Cluster Properties” on page 173](#).

After setting all of the broker’s needed JDBC configuration properties, you must also install your JDBC driver’s `.jar` file in the appropriate directory location, depending on your operating-system platform (as listed in [Appendix A, “Platform-Specific Locations of Message Queue Data”](#)) and then create the database schema for the JDBC-based data store (see [“To Set Up a JDBC-Based Data Store” on page 130](#)).

Configuring a JDBC-Based Data Store

To configure a broker to use a JDBC database, you set JDBC-related properties in the broker’s instance configuration file and create the appropriate database schema. The Message Queue Database Manager utility (`imqdbmgr`) uses your JDBC driver and the broker configuration properties to create the schema and manage the database. You can also use the Database Manager to delete corrupted tables from the database or if you want to use a different database as a data store. See [“Database Manager Utility” on page 294](#) for more information.

Note – If you use an embedded database, it is best to create it under the following directory:

```
.../instances/instanceName/dbstore/databaseName
```

If an embedded database is not protected by a user name and password, it is probably protected by file system permissions. To ensure that the database is readable and writable by the broker, the user who runs the broker should be the same user who created the embedded database using the `imqdbmgr` command.

▼ To Set Up a JDBC-Based Data Store

- 1 **Set JDBC-related properties in the broker’s instance configuration file.**

The relevant properties are discussed, with examples, in [“JDBC-Based Persistence Properties” on page 128](#) and listed in full in [Table 16–6](#). In particular, you must specify a JDBC-based data store by setting the broker’s `imq.persist.store` property to `jdbc`.

- 2 **Place a copy of, or a symbolic link to, your JDBC driver’s `.jar` file in the Message Queue external resource files directory, depending on your platform (see [Appendix A, “Platform-Specific Locations of Message Queue Data”](#)):**

Solaris: `/usr/share/lib/imq/ext`

Linux: `/opt/sun/mq/share/lib/ext`

Windows: `IMQ_VARHOME\lib\ext`

For example, if you are using HADB on a Solaris system, the following command copies the driver's .jar file to the appropriate location:

```
cp /opt/SUNWhadb/4/lib/hadbjdbc4.jar /usr/share/lib/imq/ext
```

The following command creates a symbolic link instead:

```
ln -s /opt/SUNWhadb/4/lib/hadbjdbc4.jar /usr/share/lib/imq/ext
```

3 Create the database schema needed for Message Queue persistence.

Use the `imqdbmgr create all` command (for an embedded database) or the `imqdbmgr create tbl` command (for an external database); see [“Database Manager Utility” on page 294](#).

a. Change to the directory where the Database Manager utility resides, depending on your platform:

Solaris: `cd /usr/bin`

Linux: `cd /opt/sun/mq/bin`

Windows: `cd IMQ_HOME\bin`

b. Enter the `imqdbmgr` command:

```
imqdbmgr create all
```

▼ To Display Information About a JDBC-Based Data Store

You can display information about a JDBC-based data store using the Database Manager utility (`imqdbmgr`) as follows:

1 Change to the directory where the Database Manager utility resides, depending on your platform:

Solaris: `cd /usr/bin`

Linux: `cd /opt/sun/mq/bin`

Windows: `cd IMQ_HOME\bin`

2 Enter the `imqdbmgr` command:

```
imqdbmgr query
```

The output should resemble the following

```
dbmgr query
```

```
[04/Oct/2005:15:30:20 PDT] Using plugged-in persistent store:
version=400
brokerid=Mozart1756
```

```
database connection url=jdbc:oracle:thin:@Xhome:1521:mqdb
database user=scott
Running in standalone mode.
Database tables have already been created.
```

Securing a JDBC-Based Data Store

The persistent data store can contain, among other information, message files that are being temporarily stored. Since these messages may contain proprietary information, it is important to secure the data store against unauthorized access. This section describes how to secure data in a JDBC-based data store.

A broker using JDBC-based persistence writes persistent data to a JDBC-compliant database. For a database managed by a database server (such as Oracle), it is recommended that you create a user name and password to access the Message Queue database tables (tables whose names start with MQ). If the database does not allow individual tables to be protected, create a dedicated database to be used only by Message Queue brokers. See the documentation provided by your database vendor for information on how to create user name/password access.

The user name and password required to open a database connection by a broker can be provided as broker configuration properties. However it is more secure to provide them as command line options when starting up the broker, using the `imqbrokerd` command's `-dbuserand` `-dbpassword` options (see [“Broker Utility” on page 278](#)).

For an embedded database that is accessed directly by the broker by means of the database's JDBC driver, security is usually provided by setting file permissions on the directory where the persistent data will be stored, as described above under [“Securing a File-Based Data Store” on page 128](#). To ensure that the database is readable and writable by both the broker and the Database Manager utility, however, both should be run by the same user.

Data Store Formats

Changes in the file formats for both file-based and JDBC-based persistent data stores were introduced in Message Queue 3.7, with further JDBC changes in version 4.0 and 4.1. As a result of these changes, the persistent data store version numbers have been updated to 370 for file-based data stores and 410 for JDBC-based stores. You can use the `imqdbmgr` query command to determine the version number of your existing data store.

On first startup, the Message Queue Broker utility (`imqbrokerd`) will check for the presence of an older persistent data store and automatically migrate it to the latest format:

- File-based data store versions 200 and 350 are migrated to the version 370 format.
- JDBC-based data store versions 350, 370, and 400 are migrated to the version 410 format. (If you need to upgrade a version 200 data store, you will need to step through an intermediate Message Queue 3.5 or 3.6 release.)

The upgrade leaves the older copy of the persistent data store intact, allowing you to roll back the upgrade if necessary. To do so, you can uninstall the current version of Message Queue and reinstall the earlier version you were previously running. The older version's message brokers will locate and use the older copy of the data store.

Configuring and Managing Security Services

This chapter describes Message Queue's facilities for security-related administration tasks, such as configuring user authentication, defining access control, configuring a Secure Socket Layer (SSL) connection service to encrypt client-broker communication, and setting up a password file for administrator account passwords. In addition to Message Queue's own built-in authentication mechanisms, you can also plug in a preferred external service based on the Java Authentication and Authorization Service (JAAS) API.

This chapter includes the following sections:

- “Introduction to Security Services” on page 135
- “User Authentication” on page 139
- “User Authorization” on page 153
- “Message Encryption” on page 159
- “Password Files” on page 168
- “Connecting Through a Firewall” on page 169

Introduction to Security Services

Message Queue provides security services for user access control (authentication and authorization) and for encryption:

- *Authentication* ensures that only verified users can establish a connection to a broker.
- *Authorization* specifies which users or groups have the right to access resources and to perform specific operations.
- *Encryption* protects messages from being tampered with during delivery over a connection.

As a Message Queue administrator, you are responsible for setting up the information the broker needs to authenticate users and authorize their actions. The broker properties pertaining to security services are listed under “[Security Properties](#)” on page 310. The boolean property `imq.accesscontrol.enabled` acts as a master switch that controls whether access control is

applied on a brokerwide basis; for finer control, you can override this setting for a particular connection service by setting the `imq.serviceName.accesscontrol.enabled` property, where `serviceName` is the name of the connection service, as shown in [Table 6-1](#): for example, `imq.httpjms.accesscontrol.enabled`.

The following figure shows the components used by the broker to provide authentication and authorization services. These services depend on a *user repository* containing information about the users of the messaging system: their names, passwords, and group memberships. In addition, to authorize specific operations for a user or group, the broker consults an *access control file* that specifies which operations a user or group can perform. You can designate a single access control file for the broker as a whole, using the configuration property `imq.accesscontrol.file.filename`, or for a single connection service with `imq.serviceName.accesscontrol.file.filename`.

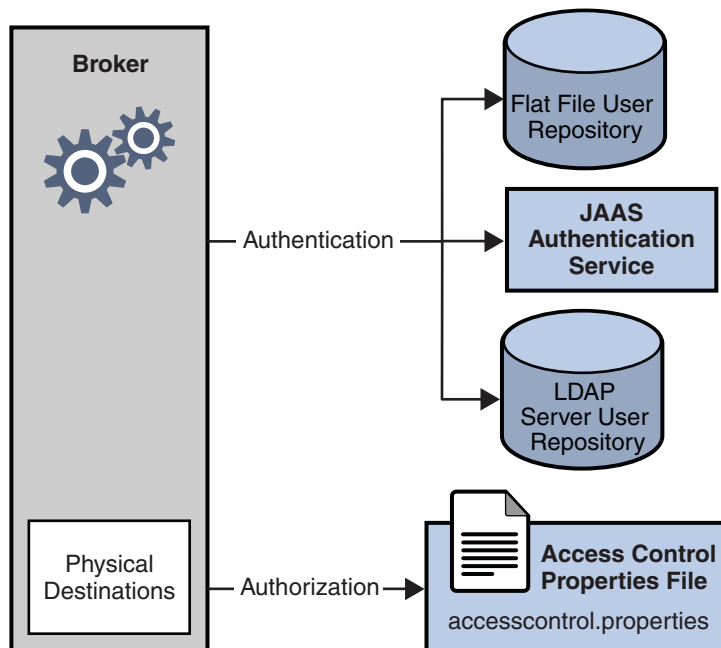


FIGURE 9-1 Security Support

As [Figure 9-1](#) shows, you can store user data in a flat file user repository that is provided with the Message Queue service, you can access an existing LDAP repository, or you can plug in a Java Authentication and Authorization Service (JAAS) module.

- If you choose a flat-file repository, you must use the `imqusermgr` utility to manage the repository. This option is easy to use and built-in.

- If you want to use an existing LDAP server, you use the tools provided by the LDAP vendor to populate and manage the user repository. You must also set properties in the broker instance configuration file to enable the broker to query the LDAP server for information about users and groups.

The LDAP option is better if scalability is important or if you need the repository to be shared by different brokers. This might be the case if you are using broker clusters.

- If you want to plug-in an existing JAAS authentication service, you need to set the corresponding properties in the broker instance configuration file.

The broker's `imq.authentication.basic.user_repository` property specifies which type of repository to use. In general, an LDAP repository or JAAS authentication service is preferable if scalability is important or if you need the repository to be shared by different brokers (if you are using broker clusters, for instance). See [“User Authentication” on page 139](#) for more information on setting up a flat-file user repository, LDAP access, or JAAS authentication service.

Authentication

A client requesting a connection to a broker must supply a user name and password, which the broker compares with those stored in the user repository. Passwords transmitted from client to broker are encoded using either base-64 encoding (for flat-file repositories) or message digest (MD5) hashing (for LDAP repositories). The choice is controlled by the `imq.authentication.type` property for the broker as a whole, or by `imq.serviceName.authentication.type` for a specific connection service. The `imq.authentication.client.response.timeout` property sets a timeout interval for authentication requests.

As described under [“Password Files” on page 168](#), you can choose to put your passwords in a *password file* instead of being prompted for them interactively. The boolean broker property `imq.passfile.enabled` controls this option. If this property is true, the `imq.passfile.dirpath` and `imq.passfile.name` properties give the directory path and file name for the password file. The `imq.imqcmd.password` property (which can be embedded in the password file) specifies the password for authenticating an administrative user to use the Command utility (`imqcmd`) for managing brokers, connection services, connections, physical destinations, durable subscriptions, and transactions.

If you are using an LDAP-based user repository, there are a whole range of broker properties available for configuring various aspects of the LDAP lookup. The address (host name and port number) of the LDAP server itself is specified by `imq.user_repository.ldap.server`. The `imq.user_repository.ldap.principal` property gives the distinguished name for binding to the LDAP repository, while `imq.user_repository.ldap.password` supplies the associated password. Other properties specify the directory bases and optional JNDI filters for individual user and group searches, the provider-specific attribute identifiers for user and group names, and so forth; see [“Security Properties” on page 310](#) for details.

Authorization

Once authenticated, a user can be authorized to perform various Message Queue-related activities. As a Message Queue administrator, you can define user groups and assign individual users membership in them. The default access control file explicitly refers to only one group, `admin` (see [“User Groups and Status” on page 139](#)). A user in this group has connection permission for the `admin` connection service, which allows the user to perform administrative functions such as creating destinations and monitoring and controlling a broker. A user in any other group that you define cannot, by default, get an `admin` service connection.

When a user attempts to perform an operation, the broker checks the user’s name and group membership (from the user repository) against those specified for access to that operation (in the access control file). The access control file specifies permissions to users or groups for the following operations:

- Connecting to a broker
- Accessing destinations: creating a consumer, a producer, or a queue browser for any given destination or for all destinations
- Auto-creating destinations

For information on configuring authorization, see [“User Authorization” on page 153](#)

Encryption

To encrypt messages sent between clients and broker, you need to use a connection service based on the Secure Socket Layer (SSL) standard. SSL provides security at the connection level by establishing an encrypted connection between an SSL-enabled broker and client.

To use an SSL-based Message Queue connection service, you generate a public/private key pair using the Message Queue Key Tool utility (`imqkeytool`). This utility embeds the public key in a self-signed certificate and places it in a Message Queue key store. The key store is itself password-protected; to unlock it, you must provide a key store password at startup time, specified by the `imq.keystore.password` property. Once the key store is unlocked, a broker can pass the certificate to any client requesting a connection. The client then uses the certificate to set up an encrypted connection to the broker.

For information on configuring encryption, see [“Message Encryption” on page 159](#)

User Authentication

Users attempting to connect to a Message Queue message broker must provide a user name and password for authentication. The broker will grant the connection only if the name and password match those in a broker-specific *user repository* listing the authorized users and their passwords. Each broker instance can have its own user repository, which you as an administrator are responsible for maintaining. This section tells how to create, populate, and manage the user repository.

Message Queue can support any of three types of authentication mechanism:

- A **flat-file repository** that is shipped with Message Queue. This type of repository is very easy to populate and manage, using the Message Queue User Manager utility (`imqusermgr`). See [“Using a Flat-File User Repository” on page 139](#).
- A **Lightweight Directory Access Protocol (LDAP) server**. This could be a new or existing LDAP directory server using the LDAP v2 or v3 protocol. You use the tools provided by the LDAP vendor to populate and manage the user repository. This type of repository is not as easy to use as the flat-file repository, but it is more scalable and therefore better for production environments. See [“Using an LDAP User Repository” on page 145](#).
- An external authentication mechanism plugged into Message Queue by means of the Java Authentication and Authorization Service (JAAS) API. See [“Using JAAS-Based Authentication” on page 148](#).

Using a Flat-File User Repository

Message Queue provides a built-in flat-file user repository and a command line tool, the User Manager utility (`imqusermgr`), for populating and managing it. Each broker has its own flat-file user repository, created automatically when you start the broker. The user repository resides in a file named `passwd`, in a directory identified by the name of the broker instance with which the repository is associated:

```
.../instances/instanceName/etc/passwd
```

(See [Appendix A, “Platform-Specific Locations of Message Queue Data,”](#) for the exact location of the instances directory, depending on your operating system platform.)

User Groups and Status

Each user in the repository can be assigned to a *user group*, which defines the default access privileges granted to all of its members. You can then specify *authorization rules* to further restrict these access privileges for specific users, as described in [“User Authorization” on page 153](#). A user’s group is assigned when the user entry is first created, and cannot be changed thereafter. The only way to reassign a user to a different group is to delete the original user entry and add another entry specifying the new group.

The flat-file user repository provides three predefined groups:

admin	For broker administrators. By default, users in this group are granted the access privileges needed to configure, administer, and manage message brokers.
user	For normal (non-administrative) client users. Newly created user entries are assigned to this group unless otherwise specified. By default, users in this group can connect to all Message Queue connection services of type <code>NORMAL</code> , produce messages to or consume messages from all physical destinations, and browse messages in any queue.
anonymous	For Message Queue clients that do not wish to use a user name known to the broker (for instance, because they do not know of a real user name to use). This group is analogous to the anonymous account provided by most FTP servers. No more than one user at a time can be assigned to this group. You should restrict the access privileges of this group in comparison to the user group, or remove users from the group at deployment time.

You cannot rename or delete these predefined groups or create new ones.

In addition to its group, each user entry in the repository has a *user status*: either *active* or *inactive*. New user entries added to the repository are marked active by default. Changing a user's status to inactive rescinds all of that user's access privileges, making the user unable to open new broker connections. Such inactive entries are retained in the user repository, however, and can be reactivated at a later time. If you attempt to add a new user with the same name as an inactive user already in the repository, the operation will fail; you must either delete the inactive user entry or give the new user a different name.

To allow the broker to be used immediately after installation without further intervention by the administrator, the flat-file user repository is created with two initial entries, summarized in [Table 9-1](#):

- The `admin` entry (user name and password `admin/admin`) enables you to administer the broker with Command utility (`imqcmd`) commands. Immediately on installation, you should update this initial entry to change its password (see [“Changing a User's Password” on page 143](#)).
- The `guest` entry allows clients to connect to the broker using a default user name and password (`guest/guest`).

You can then proceed to add any additional user entries you need for individual users of your message service.

TABLE 9-1 Initial Entries in Flat-File User Repository

User Name	Password	Group	Status
admin	admin	admin	Active
guest	guest	anonymous	Active

Using the User Manager Utility

The Message Queue User Manager utility (`imqusermgr`) enables you to populate or edit a flat-file user repository. See [“User Manager Utility” on page 295](#) for general reference information about the syntax, subcommands, and options of the `imqusermgr` command.

User Manager Preliminaries

Before using the User Manager, keep the following things in mind:

- The `imqusermgr` command must be run on the host where the broker is installed.
- If a broker-specific user repository does not yet exist, you must start up the corresponding broker instance to create it.
- You must have appropriate permissions to write to the repository; in particular, on Solaris and Linux platforms, you must be logged in as the root user or the user who first created the broker instance.

Subcommands and General Options

[Table 9-2](#) lists the subcommands of the `imqusermgr` command. For full reference information about these subcommands, see [Table 15-15](#).

TABLE 9-2 User Manager Subcommands

Subcommand	Description
add	Add user and password to repository
delete	Delete user from repository
update	Set user's password or active status (or both)
list	Display user information

The general options listed in [Table 9-3](#) apply to all subcommands of the `imqusermgr` command.

TABLE 9-3 General User Manager Options

Option	Description
-f	Perform action without user confirmation
-s	Silent mode (no output displayed)
-v	Display version information ¹
-h	Display usage help ¹

¹ Any other options specified on the command line are ignored.

Displaying the Product Version

To display the Message Queue product version, use the -v option. For example:

```
imqusermgr -v
```

If you enter an `imqusermgr` command line containing the -v option in addition to a subcommand or other options, the User Manager utility processes only the -v option. All other items on the command line are ignored.

Displaying Help

To display help on the `imqusermgr` command, use the -h option, and do not use a subcommand. You cannot get help about specific subcommands.

For example, the following command displays help about `imqusermgr`:

```
imqusermgr -h
```

If you enter an `imqusermgr` command line containing the -h option in addition to a subcommand or other options, the Command utility processes only the -h option. All other items on the command line are ignored.

Adding a User to the Repository

The subcommand `imqusermgr add` adds an entry to the user repository, consisting of a user name and password:

```
imqusermgr add [-i brokerName]
               -u userName -p password
               [-g group]
```

The -u and -p options specify the user name and password, respectively, for the new entry. These must conform to the following conventions:

- All user names and passwords must be at least one character long. Their maximum length is limited only by command shell restrictions on the maximum number of characters that can be entered on a command line.

- A user name cannot contain an asterisk (*), a comma (,), a colon (:), or a new-line or carriage-return character.
- If a user name or password contains a space, the entire name or password must be enclosed in quotation marks (" ").

The optional `-g` option specifies the group (admin, user, or anonymous) to which the new user belongs; if no group is specified, the user is assigned to the user group by default. If the broker name (`-i` option) is omitted, the default broker `imqbroker` is assumed.

For example, the following command creates a user entry on broker `imqbroker` for a user named `AliBaba`, with password `Sesame`, in the `admin` group:

```
imqusermgr add -u AliBaba -p Sesame -g admin
```

Deleting a User From the Repository

The subcommand `imqusermgr delete` deletes a user entry from the repository:

```
imqusermgr delete [-i brokerName]
                  -u userName
```

The `-u` option specifies the user name of the entry to be deleted. If the broker name (`-i` option) is omitted, the default broker `imqbroker` is assumed.

For example, the following command deletes the user named `AliBaba` from the user repository on broker `imqbroker`:

```
imqusermgr delete -u AliBaba
```

Changing a User's Password

You can use the subcommand `imqusermgr update` to change a user's password:

```
imqusermgr update [-i brokerName]
                  -u userName -p password
```

The `-u` identifies the user; `-p` specifies the new password. If the broker name (`-i` option) is omitted, the default broker `imqbroker` is assumed.

For example, the following command changes the password for user `AliBaba` to `Shazam` on broker `imqbroker`:

```
imqusermgr update -u AliBaba -p Shazam
```

Note – For the sake of security, you should change the password of the admin user from its initial default value (admin) to one that is known only to you. The following command changes the default administrator password for broker mybroker to veeblefetzer:

```
imqusermgr update -i mybroker -u admin -p veeblefetzer
```

You can quickly confirm that this change is in effect by running any of the command line tools when the broker is running. For example, the following command will prompt you for a password:

```
imqcmd list svc mybroker -u admin
```

Entering the new password (veeblefetzer) should work; the old password should fail.

After changing the password, you should supply the new password whenever you use any of the Message Queue administration tools, including the Administration Console.

Activating or Deactivating a User

The `imqusermgr update` subcommand can also be used to change a user's active status:

```
imqusermgr update [-i brokerName]  
                  -u userName -a activeStatus
```

The `-u` identifies the user; `-a` is a boolean value specifying the user's new status as active (`true`) or inactive (`false`). If the broker name (`-i` option) is omitted, the default broker `imqbroker` is assumed.

For example, the following command sets user AliBaba's status to inactive on broker `imqbroker`:

```
imqusermgr update -u AliBaba -a false
```

This renders AliBabe unable to open new broker connections.

You can combine the `-p` (password) and `-a` (active status) options in the same `imqusermgr update` command. The options may appear in either order: for example, both of the following commands activate the user entry for AliBaba and set the password to `plugh`:

```
imqusermgr update -u AliBaba -p plugh -a true  
imqusermgr update -u AliBaba -a true -p plugh
```

Viewing User Information

The `imqusermgr list` command displays information about a user in the user repository:

```
imqusermgr list [-i brokerName]  
               [-u userName]
```


The command

```
imqusermgr list -u AliBaba
```

displays information about user AliBaba, as shown in [Example 9–1](#).

EXAMPLE 9–1 Viewing Information for a Single User

```
User repository for broker instance: imqbroker
-----
User Name      Group      Active State
-----
AliBaba        admin      true
```

If you omit the `-u` option

```
imqusermgr list
```

the command lists information about all users in the repository, as in [Example 9–2](#).

EXAMPLE 9–2 Viewing Information for All Users

```
User repository for broker instance: imqbroker
-----
User Name      Group      Active State
-----
admin          admin      true
guest          anonymous  true
AliBaba        admin      true
testuser1      user       true
testuser2      user       true
testuser3      user       true
testuser4      user       false
testuser5      user       false
```

Using an LDAP User Repository

You configure a broker to use an LDAP directory server by setting the values for certain configuration properties in the broker's instance configuration file (`config.properties`). These properties enable the broker instance to query the LDAP server for information about users and groups when a user attempts to connect to the broker or perform messaging operations.

- The `imq.authentication.basic.user_repository` property specifies the kind of user authentication the broker is to use. By default, this property is set to `file`, for a flat-file user repository. For LDAP authentication, set it to `ldap` instead:

```
imq.authentication.basic.user_repository=ldap
```

- The `imq.authentication.type` property controls the type of encoding used when passing a password between client and broker. By default, this property is set to `digest`, denoting MD5 encoding, the form used by flat-file user repositories. For LDAP authentication, set it to `basic` instead:

```
imq.authentication.type=basic
```

This denotes base-64 encoding, the form used by LDAP user repositories.

- The following properties control various aspects of LDAP access. See [Table 16–8](#) for more detailed information:

```
imq.user_repository.ldap.server  
imq.user_repository.ldap.principal  
imq.user_repository.ldap.password  
imq.user_repository.ldap.propertyName  
imq.user_repository.ldap.base  
imq.user_repository.ldap.uidattr  
imq.user_repository.ldap.usrfilter  
imq.user_repository.ldap.grpsearch  
imq.user_repository.ldap.grpbase  
imq.user_repository.ldap.gidattr  
imq.user_repository.ldap.memattr  
imq.user_repository.ldap.grpfilter  
imq.user_repository.ldap.timeout  
imq.user_repository.ldap.ssl.enabled
```

- The `imq.user_repository.ldap.userformat` property, if set to a value of `dn`, specifies that the login username for authentication be in DN username format (for example: `uid=mquser,ou=People,dc=red,dc=sun,dc=com`). In this case, the broker extracts the value of the `imq.user_repository.ldap.uidattr` attribute from the DN username, and uses this value as the user name in access control operations (see [“User Authorization” on page 153](#)).
- If you want the broker to use a secure, encrypted SSL (Secure Socket Layer) connection for communicating with the LDAP server, set the broker’s `imq.user_repository.ldap.ssl.enabled` property to `true`

```
imq.user_repository.ldap.ssl.enabled=true
```

and the `imq.user_repository.ldap.server` property to the port used by the LDAP server for SSL communication: for example,

```
imq.user_repository.ldap.server=myhost:7878
```

You will also need to activate SSL communication in the LDAP server.

In addition, you may need to edit the user and group names in the broker's access control file to match those defined in the LDAP user repository; see [“User Authorization” on page 153](#) for more information.

For example, to create administrative users, you use the access control file to specify those users and groups in the LDAP directory that can create ADMIN connections.

Any user or group that can create an ADMIN connection can issue administrative commands.

▼ To Set Up an Administrative User

The following procedure makes use of a broker's access control file, which is described in [“User Authorization” on page 153](#).

1 Enable the use of the access control file by setting the broker property

`imq.accesscontrol.enabled` to `true`, which is the default value.

The `imq.accesscontrol.enabled` property enables use of the access control file.

2 Open the access control file, `accesscontrol.properties`. The location for the file is listed in [Appendix A, “Platform-Specific Locations of Message Queue Data”](#)

The file contains an entry such as the following:

```
service connection access control
#####
connection.NORMAL.allow.user=*
connection.ADMIN.allow.group=admin
```

The entries listed are examples. Note that the `admin` group exists by default in the file-based user repository but does not exist by default in the LDAP directory.

3 To grant Message Queue administrator privileges to users, enter the user names as follows:

```
connection.ADMIN.allow.user= userName[[, userName2] ...]
```

The users must be defined in the LDAP directory.

4 To grant Message Queue administrator privileges to groups, enter the group names as follows:

```
connection.ADMIN.allow.group= groupName[[, groupName2] ...]
```

The groups must be defined in the LDAP directory.

Using JAAS-Based Authentication

The Java Authentication and Authorization Service (JAAS) API allows you to plug an external authentication mechanism into Message Queue. This section describes the information that the Message Queue message broker makes available to a JAAS-compliant authentication service and explains how to configure the broker to use such a service. The following sources provide further information on JAAS:

- For complete information about the JAAS API, see the *Java™ Authentication and Authorization Service (JAAS) Reference Guide* at the URL
<http://java.sun.com/j2se/1.5.0/docs/guide/security/jaas/JAASRefGuide.html>
- For information about writing a JAAS login module, see the *Java™ Authentication and Authorization Service (JAAS) LoginModule Developer's Guide* at
<http://java.sun.com/j2se/1.5.0/docs/guide/security/jaas/JAASLMDevGuide.html>

JAAS is a core API in Java 2 Standard Edition (J2SE), and is therefore an integral part of Message Queue's runtime environment. It defines an abstraction layer between an application and an authentication mechanism, allowing the desired mechanism to be plugged in with no change to application code. In the case of the Message Queue service, the abstraction layer lies between the broker (application) and an authentication provider. By setting a few broker properties, it is possible to plug in any JAAS-compliant authentication service and to upgrade this service with no disruption or change to broker code.

Note – You cannot use the Java Management Extensions (JMX) API to change JAAS-related broker properties. However, once JAAS-based authentication is configured, JMX client applications (like other clients) can be authenticated using this mechanism.

Elements of JAAS

Figure 9–2 shows the basic elements of JAAS: a JAAS client, a JAAS-compliant authentication service, and a JAAS configuration file.

- The *JAAS client* is an application wishing to perform authentication using a JAAS-compliant authentication service. The JAAS client communicates with the authentication service using one or more *login modules* and is responsible for providing a callback handler that the login module can call to obtain the user name, password, and other information needed for authentication.
- The JAAS-compliant *authentication service* consists of one or more login modules along with logic to perform the needed authentication. The login module (`LoginModule`) may include the authentication logic itself, or it may use a private protocol or API to communicate with an external security service that provides the logic.
- The *JAAS configuration file* is a text file that the JAAS client uses to locate the login module(s) to be used.

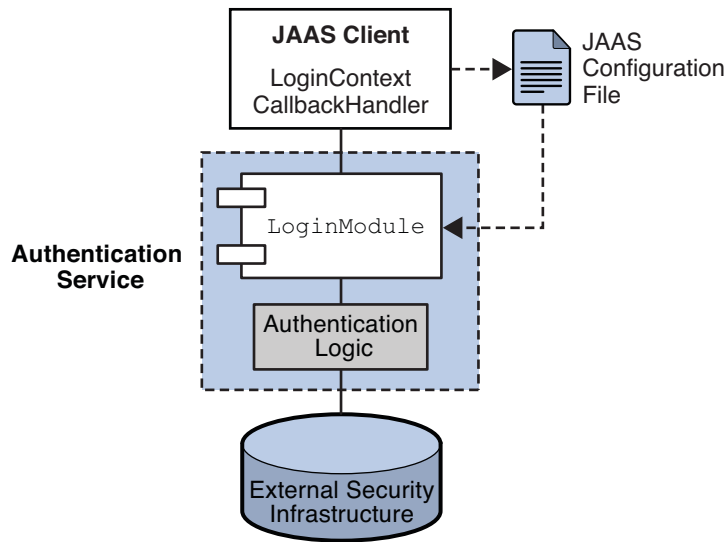


FIGURE 9-2 JAAS Elements

JAAS and Message Queue

Figure 9-3 shows how JAAS is used by the Message Queue broker. It shows a more complex implementation of the JAAS model shown in Figure 9-2.

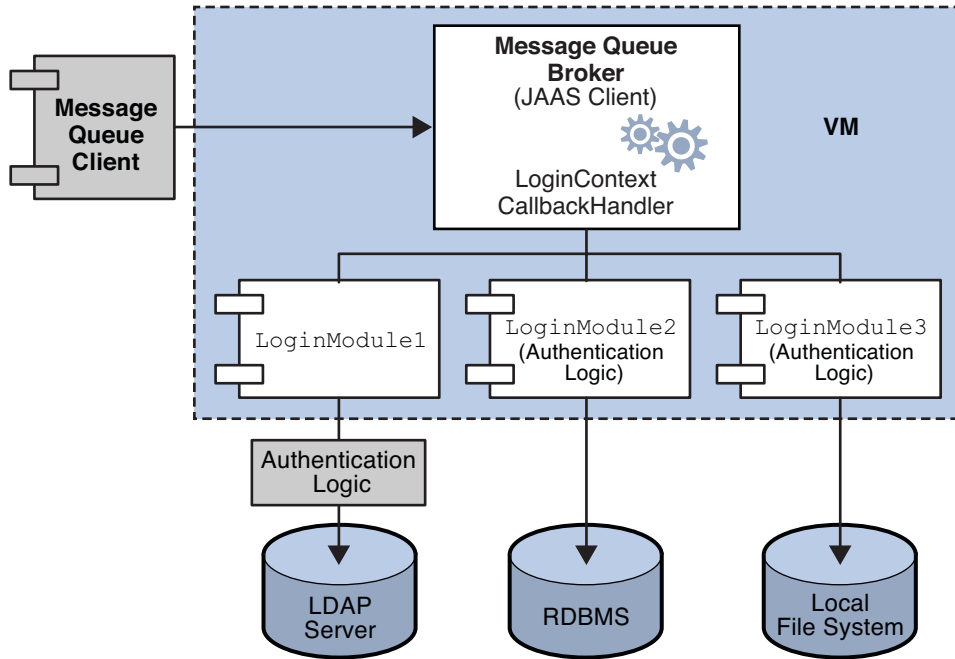


FIGURE 9-3 How Message Queue Uses JAAS

The authentication service layer, consisting of one or more login modules (if needed) and corresponding authentication logic, is separate from the broker. The login modules run in the same Java virtual machine as the broker. The broker is represented to the login module as a *login context*, and communicates with the login module by means of a *callback handler* that is part of the broker runtime code.

The authentication service also supplies a JAAS configuration file containing entries that reference the login modules. The configuration file specifies the order in which the login modules (if more than one) are to be used and any conditions for their use. When the broker starts up, it locates the configuration file by consulting either the Java system property `java.security.auth.login.config` or the Java security properties file. The broker then selects an entry in the JAAS configuration file according to the value of the broker property `imq.user_repository.jaas.name`. That entry specifies which login module(s) will be used for authentication. The classes for the login modules are found in the Message Queue external resource files directory, whose location depends on the operating system platform you are using; see [Appendix A, “Platform-Specific Locations of Message Queue Data,”](#) for details.

The relation between the configuration file, the login module, and the broker is shown in the following figure. [Figure 9-4.](#)

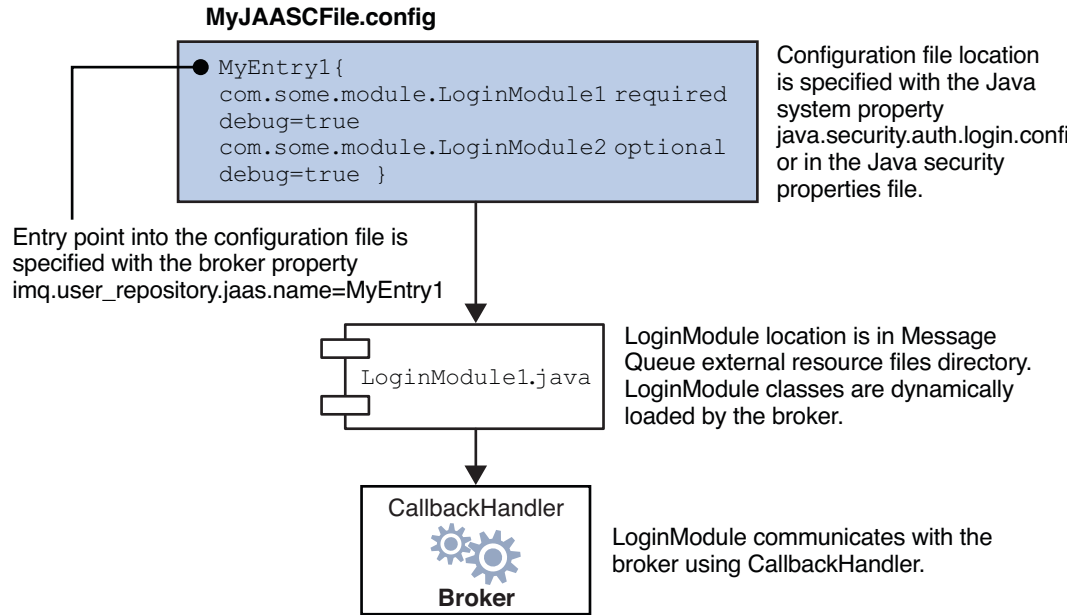


FIGURE 9-4 Setting Up JAAS Support

The fact that the broker uses a JAAS plug-in authentication service remains completely transparent to the Message Queue client. The client continues to connect to the broker as it did before, passing a user name and password. In turn, the broker uses a callback handler to pass login information to the authentication service, and the service uses the information to authenticate the user and return the results. If authentication succeeds, the broker grants the connection; if it fails, the client runtime returns a JMS security exception that the client must handle.

After the Message Queue client is authenticated, if there is further authorization to be done, the broker proceeds as it normally would, consulting the access control file to determine whether the authenticated client is authorized to perform the actions it undertakes: accessing a destination, consuming a message, browsing a queue, and so on.

Setting up JAAS-Compliant Authentication

Setting up JAAS-compliant authentication involves setting broker and system properties to select this type of authentication, to specify the location of the configuration file, and to specify the entries to the login modules that are going to be used.

To set up JAAS support for Message Queue, you perform the following general steps. (These steps assume you are creating your own authentication service.)

1. Create one or more login module classes that implement the authentication service. The JAAS callback types that the broker supports are listed below.

`javax.security.auth.callback.LanguageCallback`

The broker uses this callback to pass the authentication service the locale in which the broker is running. This value can be used for localization.

`javax.security.auth.callback.NameCallback`

The broker uses this callback to pass to the authentication service the user name specified by the Message Queue client when the connection was requested.

`javax.security.auth.callback.TextInputCallback`

The broker uses this callback to pass the value of the following information to the login module (authentication service) when requested through the `TextInputCallback.getPrompt()` with the following strings:

- `imq.authentication.type`: The broker authentication type in effect at runtime
- `imq.accesscontrol.type`: The broker access control type in effect at runtime
- `imq.authentication.clientip`: The client IP address (null if unavailable)
- `imq.servicename`: The name of the connection service (`jms`, `ssljms`, `admin`, or `ssladmin`) being used by the client
- `imq.servicetype`: The type of the connection service (`NORMAL` or `ADMIN`) being used by the client

`javax.security.auth.callback.PasswordCallback`

The broker uses this callback to pass to the authentication service the password specified by the Message Queue client when the connection was requested.

`javax.security.auth.callback.TextOutputCallback`

The broker handles this callback to provide logging service to the authentication service by logging the text output to the broker's log file. The callback's `MessageType` `ERROR`, `INFORMATION`, `WARNING` are mapped to the broker logging levels `ERROR`, `INFO`, `WARNING` respectively.

2. Create a JAAS configuration file with entries that reference the login module classes created in Step 1 and specify the location of this file.
3. Note the name of the entry in the JAAS configuration file (that references the login module implementation classes).
4. Archive the classes that implement the login modules to a jar file, and place the jar file in the Message Queue `lib/ext` directory.
5. Set the broker configuration properties that relate to JAAS support. These are described in [Table 9-4](#).
6. Set the following system property (to specify the location of the JAAS configuration file).

`java.security.auth.login.config=JAAS_Config_File_Location`

For example, you can specify the location when you start the broker.

`imqbrokerd -Djava.security.auth.login.config=JAAS_Config_File_Location`

There are other ways to specify the location of the JAAS configuration file. For additional information, please see

<http://java.sun.com/j2se/1.5.0/docs/guide/security/jaas/tutorials/LoginConfigFile.html>

The following table lists the broker properties that need to be set to set up JAAS support.

TABLE 9-4 Broker Properties for JAAS Support

Property	Description
<code>imq.authentication.type</code>	Set to <code>basic</code> to indicate Base-64 password encoding. This is the only permissible value for JAAS authentication.
<code>imq.authentication.basic.user_repository</code>	Set to <code>jaas</code> to specify JAAS authentication.
<code>imq.user_repository.jaas.name</code>	Set to the name of the desired entry (in the JAAS configuration file) that references the login modules you want to use as the authentication mechanism. This is the name you noted in Step 3.
<code>imq.user_repository.jaas.userPrincipalClass</code>	This property, used by Message Queue access control, specifies the <code>java.security.Principal</code> implementation class in the login module(s) that the broker uses to extract the Principal name to represent the user entity in the Message Queue access control file. If, it is not specified, the user name passed from the Message Queue client when a connection was requested is used instead.
<code>imq.user_repository.jaas.groupPrincipalClass</code>	This property, used by Message Queue access control, specifies the <code>java.security.Principal</code> implementation class in the login module(s) that the broker uses to extract the Principal name to represent the group entity in the Message Queue access control file. If, it is not specified, the group rules, if any, in the Message Queue access control file are ignored.

User Authorization

An *access control file* contains rules that specify which users (or groups of users) are authorized to perform certain operations on a message broker. These operations include the following:

- Creating a connection
- Creating a message producer for a physical destination
- Creating a message consumer for a physical destination
- Browsing a queue destination
- Auto-creating a physical destination

If access control is enabled (that is, if the broker's `imq.accesscontrol.enabled` configuration property is set to `true`, the broker will consult its access control file whenever a client attempts one of these operations, to verify whether the user generating the request (or a group to which the user belongs) is authorized to perform the operation. By editing this file, you can restrict access to these operations to particular users and groups. Changes take effect immediately; there is no need to restart the broker after editing the file.

Access Control File Syntax

Each broker has its own access control file, created automatically when the broker is started. The file is named `accesscontrol.properties` and is located at a path of the form

```
.../instances/brokerInstanceName/etc/accesscontrol.properties
```

(See [Appendix A](#), “Platform-Specific Locations of Message Queue Data,” for the exact location, depending on your platform.)

The file is formatted as a Java properties file. It starts with a `version` property defining the version of the file:

```
version=JMQFileAccessControlModel/100
```

This is followed by three sections specifying the access control for three categories of operations:

- Creating connections
- Creating message producers or consumers, or browsing a queue destination
- Auto-creating physical destinations

Each of these sections consists of a sequence of *authorization rules* specifying which users or groups are authorized to perform which specific operations. These rules have the following syntax:

```
resourceType.resourceVariant.operation.access.principalType=principals
```

[Table 9–5](#) describes the various elements.

TABLE 9–5 Authorization Rule Elements

Element	Description
<i>resourceType</i>	Type of resource to which the rule applies: connection: Connections queue: Queue destinations topic: Topic destinations

TABLE 9-5 Authorization Rule Elements (Continued)

Element	Description
<i>resourceVariant</i>	Specific resource (connection service type or destination) to which the rule applies An asterisk (*) may be used as a wild-card character to denote all resources of a given type: for example, a rule beginning with <code>queue.*</code> applies to all queue destinations.
<i>operation</i>	Operation to which the rule applies This syntax element is not used for <code>resourceType=connection</code> .
<i>access</i>	Level of access authorized: allow: Authorize user to perform operation deny: Prohibit user from performing operation
<i>principalType</i>	Type of principal (user or group) to which the rule applies: user: Individual user group: User group
<i>principals</i>	List of principals (users or groups) to whom the rule applies, separated by commas An asterisk (*) may be used as a wild-card character to denote all users or all groups: for example, a rule ending with <code>user=*</code> applies to all users.

EXAMPLE 9-3 Example 1

Rule: `queue.q1.consume.allow.user=*`

Description: allows all users to consume messages from the queue destination q1.

EXAMPLE 9-4 Example 2

Rule: `queue.*.consume.allow.user=Snoopy`

Description: allows user Snoopy to consume messages from all queue destinations.

EXAMPLE 9-5 Example 3

Rule: `topic.t1.produce.deny.user=Snoopy`

Description: prevents Snoopy from producing messages to the topic destination t1

Note – You can use Unicode escape (`\uXXXX`) notation to specify non-ASCII user, group, or destination names. If you have edited and saved the access control file with these names in a non-ASCII encoding, you can use the `Java native2ascii` tool to convert the file to ASCII. See the *Java Internationalization FAQ* at

<http://java.sun.com/j2se/1.4/docs/guide/intl/faq.html>

for more information.

Application of Authorization Rules

Authorization rules in the access control file are applied according to the following principles:

- Any operation not explicitly authorized through an authorization rule is implicitly prohibited. For example, if the access control file contains no authorization rules, all users are denied access to all operations.
- Authorization rules for specific users override those applying generically to all users. For example, the rules

```
queue.q1.produce.allow.user=*
queue.q1.produce.deny.user=Snoopy
```

authorize all users except Snoopy to send messages to queue destination q1.

- Authorization rules for a specific user override those for any group to which the user belongs. For example, if user Snoopy is a member of group user, the rules

```
queue.q1.consume.allow.group=user
queue.q1.consume.deny.user=Snoopy
```

authorize all members of user except Snoopy to receive messages from queue destination q1.

- Authorization rules applying generically to all users override those applying to all groups. For example, the rules

```
topic.t1.produce.deny.group=*
topic.t1.produce.allow.user=*
```

authorize all users to publish messages to topic destination t1, overriding the rule denying such access to all groups.

- Authorization rules for specific resources override those applying generically to all resources of a given type. For example, the rules

```
topic.*.consume.allow.user=Snoopy
topic.t1.consume.deny.user=Snoopy
```

authorize Snoopy to subscribe to all topic destinations except t1.

- Authorization rules authorizing and denying access to the same resource and operation for the same user or group cancel each other out, resulting in authorization being denied. For example, the rules

```
queue.q1.browse.deny.user=Snoopy
queue.q1.browse.allow.user=Snoopy
```

prevent Snoopy from browsing queue q1. The rules

```
topic.t1.consume.deny.group=user
topic.t1.consume.allow.group=user
```

prevent all members of group user from subscribing to topic t1.

- When multiple authorization rules are specified for the same resource, operation, and principal type, only the last rule applies. The rules

```
queue.q1.browse.allow.user=Snoopy, Linus
queue.q1.browse.allow.user=Snoopy
```

authorize user Snoopy, but not Linus, to browse queue destination q1.

Authorization Rules for Connection Services

Authorization rules with the resource type `connection` control access to the broker's connection services. The rule's *resourceVariant* element specifies the service type of the connection services to which the rule applies, as shown in [Table 6-1](#); the only possible values are `NORMAL` or `ADMIN`. There is no *operation* element.

The default access control file contains the rules

```
connection.NORMAL.allow.user=*
connection.ADMIN.allow.group=admin
```

giving all users access to `NORMAL` connection services (`jms`, `ssljms`, `httpjms`, and `httpsjms`) and those in the `admin` group access to `ADMIN` connection services (`admin` and `ssladmin`). You can then add additional authorization rules to restrict the connection access privileges of specific users: for example, the rule

```
connection.NORMAL.deny.user=Snoopy
```

denies user Snoopy access privileges for connection services of type `NORMAL`.

If you are using a file-based user repository, the `admin` user group is created by the User Manager utility. If access control is disabled (`imq.accesscontrol.enabled = false`), all users in the `admin` group automatically have connection privileges for `ADMIN` connection services. If access control is enabled, access to these services is controlled by the authorization rules in the access control file.

If you are using an LDAP user repository, you must define your own user groups in the LDAP directory, using the tools provided by your LDAP vendor. You can either define a group named `admin`, which will then be governed by the default authorization rule shown above, or edit the access control file to refer to one or more other groups that you have defined in the LDAP directory. You must also explicitly enable access control by setting the broker's `imq.accesscontrol.enabled` property to `true`.

Authorization Rules for Physical Destinations

Access to specific physical destinations on the broker is controlled by authorization rules with a resource type of `queue` or `topic`, as the case may be. These rules regulate access to the following operations:

- Sending messages to a queue: `produce` operation
- Receiving messages from a queue: `consume` operation
- Publishing messages to a topic: `produce` operation
- Subscribing to and consuming messages from a topic: `consume` operation
- Browsing a queue: `browse` operation

By default, all users and groups are authorized to perform all of these operations on any physical destination. You can change this by editing the default authorization rules in the access control properties file or overriding them with more specific rules of your own. For example, the rule

```
topic.Admissions.consume.deny.group=user
```

denies all members of the user group the ability to subscribe to the topic `Admissions`.

The final section of the access control file, includes authorization rules that specify for which users and groups the broker will auto-create a physical destination.

When a client creates a message producer or consumer for a physical destination that does not already exist, the broker will *auto-create* the destination (provided that the broker's `imq.autocreate.queue` or `imq.autocreate.topic` property is set to `true`).

A separate section of the access control file controls the ability of users and groups to perform such auto-creation. This is governed by authorization rules with a *resourceType* of `queue` or `topic` and an *operation* element of `create`. the *resourceVariant* element is omitted, since these rules apply to all queues or all topics, rather than any specific destination.

The default access control file contains the rules

```
queue.create.allow.user=*
topic.create.allow.user=*
```

authorizing all users to have physical destinations auto-created for them by the broker. You can edit the file to restrict such authorization for specific users. For example, the rule

```
topic.create.deny.user=Snoopy
```

denies user Snoopy the ability to auto-create topic destinations.

Note – Note that the effect of such auto-creation rules must be congruent with that of other physical destination access rules. For example, if you change the destination authorization rule to prohibit any user from sending a message to a queue, but enable the auto-creation of queue destinations, the broker *will* create the physical destination if it does not exist, but will *not* deliver a message to it.

Message Encryption

This section explains how to set up a connection service based on the Secure Socket Layer (SSL) standard, which enables delivery of encrypted messages over the connection. Message Queue supports the following SSL-based connection services:

- The `ssljms` service delivers secure, encrypted messages between a client and a broker, using the TCP/IP transport protocol.
- The `httpsjms` service delivers secure, encrypted messages between a client and a broker, using an HTTPS tunnel servlet with the HTTP transport protocol.
- The `ssladmin` service creates a secure, encrypted connection between the Message Queue Command utility (`imqcmd`) and a broker, using the TCP/IP transport protocol. Encrypted connections are not supported for the Administration Console (`imqadmin`).
- The `cluster` connection service is used internally to provide secure, encrypted communication between brokers in a cluster, using the TCP/IP transport protocol.
- A JMX connector that supports secure, encrypted communication between a JMX client and a broker's MBean server using the RMI transport protocol over TCP.

The remainder of this section describes how to set up secure connections over TCP/IP, using the `ssljms`, `ssladmin`, and `cluster` connection services. For information on setting up secure connections over HTTP with the `httpsjms` service, see [Appendix C, “HTTP/HTTPS Support.”](#)

Using Self-Signed Certificates

To use an SSL-based connection service over TCP/IP, you generate a public/private key pair using the Key Tool utility (`imqkeytool`). This utility embeds the public key in a self-signed certificate that is passed to any client requesting a connection to the broker, and the client uses the certificate to set up an encrypted connection. This section describes how to set up an SSL-based service using such self-signed certificates.

For a stronger level of authentication, you can use signed certificates verified by a certification authority. The use of signed certificates involves some additional steps beyond those needed for

self-signed certificates: you must first perform the procedures described in this section and then perform the additional steps in [“Using Signed Certificates” on page 165](#).

Message Queue's support for SSL with self-signed certificates is oriented toward securing on-the-wire data, on the assumption that the client is communicating with a known and trusted server. Configuring SSL with self-signed certificates requires configuration on both the broker and client:

- [“Setting Up an SSL-Based Connection Service Using Self-Signed Certificates” on page 160](#)
- [“Configuring and Running an SSL-Based Client Using Self-Signed Certificates” on page 164](#)

Setting Up an SSL-Based Connection Service Using Self-Signed Certificates

The following sequence of procedures are needed to set up an SSL-based connection service for using self-signed certificates:

Note – Starting with release 4.0, the default value for the client connection factory property `imqSSLIsHostTrusted` is `false`. If your application depends on the prior default value of `true`, you need to reconfigure and to set the property explicitly to `true`. In particular, old or new clients using self-signed certificates should set this property to `true`; for example:

```
java -DimqConnectionType=TLS -DimqSSLIsHostTrusted=true MyApp
```

The administration tool `imqcmd` is also affected by this change. In addition to using the `-secure` option to specify that it uses a SSL-based admin connection service, the `imqSSLIsHostTrusted` should be set to `true` when connecting to a broker configured with a self-signed certificate. You can do this as follows:

```
imqcmd list svc -secure -DimqSSLIsHostTrusted=true
```

Alternatively, you can import the broker's self-signed certificate into the client runtime trust store. Use the procedure in [“To Install a Signed Certificate” on page 166](#).

1. Generate a self-signed certificate.
2. Enable the desired SSL-based connection services in the broker. These can include the `ssljms`, `ssladmin`, or `cluster` connection services.
3. Start the broker.

▼ To Generate a Self-Signed Certificate

Run the Key Tool utility (`imqkeytool`) to generate a self-signed certificate for the broker. (On Solaris and Linux operating systems, you may need to run the utility as the root user in order to have permission to create the keystore file.) The same certificate can be used for all SSL-based connection services (`ssljms`, `ssladmin`, `cluster` connection services, and the `ssljmxrmi` connector).

1 Enter the following at the command prompt:

```
imqkeytool -broker
```

The Key Tool utility prompts you for a key store password:

2 At the prompt type a keystore password.

The Keystore utility prompts you for identifying information from which to construct an X.500 distinguished name. The following table shows the prompts and the values to be provided for each. Values are case-insensitive and can include spaces.

Prompt	X.500 Attribute	Description	Example
What is your first and last name?	commonName (CN)	Fully qualified name of server running the broker	mqserver.sun.com
What is the name of your organizational unit?	organizationalUnit (OU)	Name of department or division	purchasing
What is the name of your organization?	organizationName (ON)	Name of larger organization, such as a company or government entity	Acme Widgets, Inc.
What is the name of your city or locality?	localityName (L)	Name of city or locality	San Francisco
What is the name of your state or province?	stateName (ST)	Full (unabbreviated) name of state or province	California
What is the two-letter country code for this unit?	country (C)	Standard two-letter country code	US

The Key Tool utility displays the information you entered for confirmation. For example,

```
Is CN=mqserver.sun.com, OU=purchasing, ON=Acme Widgets, Inc.,
L=San Francisco, ST=California, C=US correct?
```

3 Accept the current values and proceed by typing yes.

To reenter values, accept the default or enter no. After you confirm, the utility pauses while it generates a key pair.

The utility asks for a password to lock the key pair (key password).

4 Press return.

This will set the same password for both the key password and the keystore password.



Caution – Be sure to remember the password you specify. You must provide this password when you start the broker, to allow the broker to open the keystore file. You can store the keystore password in a password file (see [“Password Files” on page 168](#)).

The Key Tool utility generates a self-signed certificate and places it in Message Queue’s keystore file. The keystore file is located in a directory whose location depends upon the operating system platform, as shown in [Appendix A, “Platform-Specific Locations of Message Queue Data.”](#)

The following are the configurable properties for the Message Queue keystore for SSL-based connection services:

<code>imq.keystore.file.dirpath</code>	Path to directory containing keystore file (see Appendix A, “Platform-Specific Locations of Message Queue Data,” for default value)
<code>imq.keystore.file.name</code>	Name of key store file
<code>imq.keystore.password</code>	Ke store password (to be used only in a password file)

In some circumstances, you may need to regenerate a key pair in order to solve certain problems: for example, if you forget the key store password or if the SSL-based service fails to initialize when you start a broker and you get the exception:

```
java.security.UnrecoverableKeyException: Cannot recover key
```

(This exception may result if you provided a key password different from the keystore password when you generated the self-signed certificate.)

▼ To Regenerate a Key Pair

1 Remove the broker’s keystore file.

The file is located as shown in [Appendix A, “Platform-Specific Locations of Message Queue Data.”](#)

2 Run `imqkeytool` again.

The command will generate a new key pair, as described above.

▼ To Enable an SSL-Based Connection Service in the Broker

To enable an SSL-based connection service in the broker, you need to add the corresponding service or services to the `imq.service.activelist` property.

1 Open the broker's instance configuration file.

The instance configuration file is located in a directory identified by the name of the broker instance (*instanceName*) with which the configuration file is associated (see [Appendix A, “Platform-Specific Locations of Message Queue Data”](#)):

```
.../instances/instanceName/props/config.properties
```

2 Add an entry (if one does not already exist) for the `imq.service.activelist` property and include the desired SSL-based service(s) in the list.

By default, the property includes the `jms` and `admin` connection services. Add the SSL-based service or services you wish to activate (`ssljms`, `ssladmin`, or both):

```
imq.service.activelist=jms,admin,ssljms,ssladmin
```

Note – The SSL-based cluster connection service is enabled using the `imq.cluster.transport` property rather than the `imq.service.activelist` property (see [“Cluster Connection Service Properties” on page 172](#)). To enable SSL for RMI-based JMX connectors, see [“SSL-Based JMX Connections” on page 406](#).

3 Save and close the instance configuration file.

▼ To Start the Broker

Start the broker, providing the key store password.

Note – When you start a broker or client with SSL, you may notice a sharp increase in CPU usage for a few seconds. This is because the JSSE (Java Secure Socket Extension) method `java.security.SecureRandom`, which Message Queue uses to generate random numbers, takes a significant amount of time to create the initial random number seed. Once the seed is created, the CPU usage level will drop to normal.

1 Start the broker, providing the keystore password.

Put the keystore password in a password file, as described in [“Password Files” on page 168](#) and set the `imq.passfile.enabled` property to `true`. You can now do one of the following:

- **Pass the location of the password file to the `imqbrokerd` command:**

```
imqbrokerd -passfile /passfileDirectory/passfileName
```

- **Start the broker without the `-passfile` option, but specify the location of the password file using the following two broker configuration properties:**

```
imq.passfile.dirpath=/passfileDirectory
```

```
imq.passfile.name=/passfileName
```

- 2 If you are not using a password file, enter the keystore password at the prompt.

```
imqbrokerd
```

You are prompted for the keystore password.

Configuring and Running an SSL-Based Client Using Self-Signed Certificates

The procedure for configuring a client to use an SSL-based connection service differs depending on whether it is an application client (using the `ssljms` connection service) or a Message Queue administrative client such as `imqcmd` (using the `ssladmin` connection service.)

Application Clients

For application clients, you must make sure the client has the following `.jar` files specified in its `CLASSPATH` variable:

```
imq.jar
jms.jar
```

Once the `CLASSPATH` files are properly specified, one way to start the client and connect to the broker's `ssljms` connection service is by entering a command like the following:

```
java -DimqConnectionType=TLS clientAppName
```

This tells the connection to use an SSL-based connection service.

Administrative Clients

For administrative clients, you can establish a secure connection by including the `-secure` option when you invoke the `imqcmd` command: for example,

```
imqcmd list svc -b hostName:portNumber -u userName -secure
```

where *userName* is a valid ADMIN entry in the Message Queue user repository. The command will prompt you for the password.

Listing the connection services is a way to verify that the `ssladmin` service is running and that you can successfully make a secure administrative connection, as shown in [Example 9–6](#).

EXAMPLE 9–6 Connection Services Listing

Listing all the services on the broker specified by:

Host	Primary Port	
localhost	7676	

Service Name	Port Number	Service State
admin	33984 (dynamic)	RUNNING
httpjms	-	UNKNOWN
httpsjms	-	UNKNOWN
jms	33983 (dynamic)	RUNNING
ssladmin	35988 (dynamic)	RUNNING
ssljms	dynamic	UNKNOWN

Successfully listed services.

Using Signed Certificates

Signed certificates provide a stronger level of server authentication than self-signed certificates. You can implement signed certificates only between a client and broker, and currently not between multiple brokers in a cluster. This requires the following extra procedures in addition to the ones described in [“Using Self-Signed Certificates” on page 159](#). Using signed certificates requires configuration on both the broker and client:

- [“Obtaining and Installing a Signed Certificate” on page 165](#)
- [“Configuring the Client to Require Signed Certificates” on page 167](#)

Obtaining and Installing a Signed Certificate

The following procedures explain how to obtain and install a signed certificate.

▼ To Obtain a Signed Certificate

- 1 Use the `J2SE keytool` command to generate a certificate signing request (CSR) for the self-signed certificate you generated in the preceding section.

Information about the `keytool` command can be found at

<http://java.sun.com/j2se/1.5.0/docs/tooldocs/solaris/keytool.html>

Here is an example:

```
keytool -certreq -keyalg RSA -alias imq -file certreq.csr
        -keystore /etc/imq/keystore -storepass myStorePassword
```

This generates a CSR encapsulating the certificate in the specified file (certreq.csr in the example).

2 Use the CSR to generate or request a signed certificate.

You can do this by either of the following methods:

- Have the certificate signed by a well known certification authority (CA), such as Thawte or Verisign. See your CA's documentation for more information on how to do this.
- Sign the certificate yourself, using an SSL signing software package.

The resulting signed certificate is a sequence of ASCII characters. If you receive the signed certificate from a CA, it may arrive as an e-mail attachment or in the text of a message.

3 Save the signed certificate in a file.

The instructions below use the example name broker.cer to represent the broker certificate.

▼ To Install a Signed Certificate

1 Check whether J2SE supports your certification authority by default.

The following command lists the root CAs in the system key store:

```
keytool -v -list -keystore $JAVA_HOME/lib/security/cacerts
```

If your CA is listed, skip the next step.

2 If your certification authority is not supported in J2SE, import the CA's root certificate into the Message Queue key store.

Here is an example:

```
keytool -import -alias ca -file ca.cer -noprompt -trustcacerts
        -keystore /etc/imq/keystore -storepass myStorePassword
```

where ca.cer is the file containing the root certificate obtained from the CA.

If you are using a CA test certificate, you probably need to import the test CA root certificate. Your CA should have instructions on how to obtain a copy.

3 Import the signed certificate into the key store to replace the original self-signed certificate.

Here is an example:

```
keytool -import -alias imq -file broker.cer -noprompt -trustcacerts
        -keystore /etc/imq/keystore -storepass myStorePassword
```

where broker.cer is the file containing the signed certificate that you received from the CA.

The Message Queue key store now contains a signed certificate to use for SSL connections.

Configuring the Client to Require Signed Certificates

You must now configure the Message Queue client runtime to require signed certificates, and ensure that it trusts the certification authority that signed the certificate.

Note – By default, starting with release 4.0, the connection factory object that the client will be using to establish broker connections has its `imqSSLIsHostTrusted` attribute set to `false`, meaning that the client runtime will attempt to validate all certificates. Validation will fail if the signer of the certificate is not in the client's trust store.

▼ To Configure the Client Runtime to Require Signed Certificates

1 Verify whether the signing authority is registered in the client's trust store.

To test whether the client will accept certificates signed by your certification authority, try to establish an SSL connection, as described above under [“Configuring and Running an SSL-Based Client Using Self-Signed Certificates” on page 164](#). If the CA is in the client's trust store, the connection will succeed and you can skip the next step. If the connection fails with a certificate validation error, go on to the next step.

2 Install the signing CA's root certificate in the client's trust store.

The client searches the key store files `cacerts` and `jssecacerts` by default, so no further configuration is necessary if you install the certificate in either of those files. The following example installs a test root certificate from the Verisign certification authority from a file named `testrootca.cer` into the default system certificate file, `cacerts`. The example assumes that J2SE is installed in the directory `$JAVA_HOME/usr/j2se`:

```
keytool -import -keystore /usr/j2se/jre/lib/security/cacerts
        -alias VerisignTestCA -file testrootca.cer -noprompt
        -trustcacerts -storepass myStorePassword
```

An alternative (and recommended) option is to install the root certificate into the alternative system certificate file, `jssecacerts`:

```
keytool -import -keystore /usr/j2se/jre/lib/security/jssecacerts
        -alias VerisignTestCA -file testrootca.cer -noprompt
        -trustcacerts -storepass myStorePassword
```

A third possibility is to install the root certificate into some other key store file and configure the client to use that as its trust store. The following example installs into the file `/home/smith/.keystore`:

```
keytool -import -keystore /home/smith/.keystore
        -alias VerisignTestCA -file testrootca.cer -noprompt
        -trustcacerts -storepass myStorePassword
```

Since the client does not search this key store by default, you must explicitly provide its location to the client to use as a trust store. You do this by setting the Java system property `javax.net.ssl.trustStore` once the client is running:

```
javax.net.ssl.trustStore=/home/smith/.keystore
```

Password Files

Several types of command require passwords. In [Table 9–6](#), the first column lists the commands that require passwords and the second column lists the reason that passwords are needed.

TABLE 9–6 Commands That Use Passwords

Command	Description	Purpose of Password
imqbrokerd	Start broker	Access a JDBC-based persistent data store, an SSL certificate key store, or an LDAP user repository
imqcmd	Manage broker	Authenticate an administrative user who is authorized to use the command
imqdbmgr	Manage JDBC-based data store	Access the data store

You can specify these passwords in a *password file* and use the `-passfile` option to specify the name of the file. This is the format for the `-passfile` option:

```
imqbrokerd -passfile filePath
```

Note – In previous versions of Message Queue, you could use the `-p`, `-password`, `-dbpassword`, and `-ldappassword` options to specify passwords on the command line. As of Message Queue 4.0, these options are deprecated and are no longer supported; you must use a password file instead.

Security Concerns

Typing a password interactively, in response to a prompt, is the most secure method of specifying a password (provided that your monitor is not visible to other people). You can also specify a password file on the command line. For non-interactive use of commands, however, you must use a password file.

A password file is unencrypted, so you must set its permissions to protect it from unauthorized access. Set the permissions so that they limit the users who can view the file, but provide read access to the user who starts the broker.

Password File Contents

A password file is a simple text file containing a set of properties and values. Each value is a password used by a command. [Table 9–7](#) shows the types of passwords that a password file can contain.

TABLE 9–7 Passwords in a Password File

Password	Affected Commands	Description
imq.imqcmd.password	imqcmd	Administrator password for Message Queue Command utility (authenticated for each command)
imq.keystore.password	imqbrokerd	Key store password for SSL-based services
imq.persist.jdbc.password	imqbrokerd imqdbmgr	Password for opening a database connection, if required
imq.user_repository.ldap.password	imqbrokerd	Password associated with the distinguished name assigned to a broker for binding to a configured LDAP user repository

A sample password file is provided as part of your Message Queue installation; see [Appendix A](#), “Platform-Specific Locations of Message Queue Data,” for the location of this file, depending on your platform.

Connecting Through a Firewall

When a client application is separated from the broker by a firewall, special measures are needed in order to establish a connection. One approach is to use the `httpjms` or `httpsjms` connection service, which can “tunnel” through the firewall; see [Appendix C](#), “HTTP/HTTPS Support,” for details. HTTP connections are slower than other connection services, however; a faster alternative is to bypass the Message Queue Port Mapper and explicitly assign a static port address to the desired connection service, and then open that specific port in the firewall. This approach can be used to connect through a firewall using the `jms` or `ssljms` connection service (or, in unusual cases, `admin` or `ssladmin`).

TABLE 9–8 Broker Configuration Properties for Static Port Addresses

Connection Service	Configuration Property
jms	imq.jms.tcp.port
ssljms	imq.ssljms.tls.port

TABLE 9–8 Broker Configuration Properties for Static Port Addresses (Continued)

Connection Service	Configuration Property
admin	imq.admin.tcp.port
ssladmin	imq.ssladmin.tls.port

▼ To Enable Broker Connections Through a Firewall

1 Assign a static port address to the connection service you wish to use.

To bypass the Port Mapper and assign a static port number directly to a connection service, set the broker configuration property `imq.serviceName.protocolType.port`, where *serviceName* is the name of the connection service and *protocolType* is its protocol type (see Table 9–8). As with all broker configuration properties, you can specify this property either in the broker's instance configuration file or from the command line when starting the broker. For example, to assign port number 10234 to the `jms` connection service, either include the line

```
imq.jms.tcp.port=10234
```

in the configuration file or start the broker with the command

```
imqbrokerd -name brokerName -Dimq.jms.tcp.port=10234
```

where *brokerName* is the name of the broker to be started.

2 Configure the firewall to allow connections to the port number you assigned to the connection service.

You must also allow connections through the firewall to Message Queue's Port Mapper port (normally 7676, unless you have reassigned it to some other port). In the example above, for instance, you would need to open the firewall for ports 10234 and 7676.

Configuring and Managing Broker Clusters

Message Queue supports the use of *broker clusters*: groups of brokers working together to provide message delivery services to clients. Clusters enable a message service to scale its operations to meet an increasing volume of message traffic by distributing client connections among multiple brokers.

In addition, clusters provide for *message service availability*. In the case of a *conventional* cluster, if a broker fails, clients connected to that broker can reconnect to another broker in the cluster and continue producing and consuming messages. In the case of a *high-availability* cluster, if a broker fails, clients connected to that broker reconnect to a failover broker that takes over the pending work of the failed broker, delivering messages without interruption of service.

See the *Message Queue Technical Overview* for a description of conventional and high-availability broker clusters and how they operate.

This chapter describes how to configure and manage both conventional and high-availability broker clusters:

- [“Configuring Broker Clusters” on page 171](#)
- [“Managing Broker Clusters” on page 178](#)

Configuring Broker Clusters

You create a broker cluster by specifying *cluster configuration properties* for each of its member brokers. Except where noted in this chapter, cluster configuration properties must be set to the same value for each broker in a cluster. This section introduces these properties and the use of a *cluster configuration file* to specify them:

- [“The Cluster Configuration File” on page 172](#)
- [“Cluster Configuration Properties” on page 172](#)
- [“Displaying a Cluster Configuration” on page 176](#)

The Cluster Configuration File

Like all broker properties, cluster configuration properties can be set individually for each broker in a cluster, either in its instance configuration file (`config.properties`) or by using the `-D` option on the command line when you start the broker. However, except where noted in this chapter, each cluster configuration property must be set to the same value for each broker in a cluster. must be the same

For example, to specify the transport protocol for the cluster connection service, you can include the following property in the instance configuration file for each broker in the cluster:

```
imq.cluster.transport=ssl
```

Notice, however, that if you need to change the value of this property, you must change its value in the instance configuration file for every broker in the cluster.

For consistency and ease of maintenance, it is generally more convenient to collect all of the common cluster configuration properties into a central cluster configuration file that all of the individual brokers in a cluster reference. Using a cluster configuration file prevents the settings from getting out of synch and ensures that all brokers in a cluster use the same, consistent configuration information.

When using a cluster configuration file, each broker's instance configuration file must point to the location of the cluster configuration file by setting the `imq.cluster.url` property. For example,

```
imq.cluster.url=file:/home/cluster.properties
```

Cluster Configuration Properties

This section reviews the most important cluster configuration properties, grouped into the following categories:

- “Cluster Connection Service Properties” on page 172
- “Conventional Cluster Properties” on page 173
- “High-Availability Cluster Properties” on page 173

A complete list of cluster configuration properties can be found in [Table 16–11](#)

Cluster Connection Service Properties

The following properties are used to configure the cluster connection service used for internal communication between brokers in the cluster. These properties are used by both conventional and high-availability clusters.

- **imq.cluster.transport** specifies the transport protocol used by the cluster connection service, such as `tcp` or `ssl`.

- **imq.cluster.port** specifies the port number for the cluster connection service. You might need to set this property, for instance, to specify a static port number for connecting to the broker through a firewall.
- **imq.cluster.hostname** specifies the host name or IP address for the cluster connection service, used for internal communication between brokers in the cluster. This setting can be useful if there is more than one network interface card installed in a computer.

Conventional Cluster Properties

The following properties, in addition to those listed in “[Cluster Connection Service Properties](#)” on page 172, are used to configure conventional clusters:

- **imq.cluster.brokerlist** specifies a list of broker addresses defining the membership of the cluster; all brokers in the cluster must have the same value for this property.
For example, to create a conventional cluster consisting of brokers at port 9876 on host1, port 5000 on host2, and the default port (7676) on ctrlhost, use the following value:
`imq.cluster.brokerlist=host1:9876,host2:5000,ctrlhost`
- **imq.cluster.masterbroker** specifies which broker in a conventional cluster is the master broker that maintains the configuration change record that tracks the addition and deletion of destinations and durable subscribers. For example:
`imq.cluster.masterbroker=host2:5000`

High-Availability Cluster Properties

High-availability broker clusters, which share a JDBC-based data store, require more configuration than do conventional broker clusters. In addition to the properties listed in “[Cluster Connection Service Properties](#)” on page 172, the following categories of properties are used to configure a high-availability cluster:

- “[High-Availability Clusters: General Configuration Properties](#)” on page 173
- “[High-Availability Clusters: JDBC Configuration Properties](#)” on page 174
- “[High-Availability Clusters: Failure Detection Properties](#)” on page 175

High-Availability Clusters: General Configuration Properties

- **imq.cluster.ha** is a boolean value that specifies if the cluster is high-availability cluster (true) or a conventional broker (false). The default value is false.
If set to true, the high-availability mechanisms for failure detection and takeover of a failed broker are enabled. High-availability clusters are self-configuring: any broker configured to use the cluster’s shared data store is automatically registered as part of the cluster, without further action on your part. If the conventional cluster property, `imq.cluster.brokerlist`, is specified for a high-availability broker, the property is ignored and a warning message is logged at broker startup.

- **imq.persist.store** specifies the model for a broker's persistent data store. This property must be set to the value `jdbc` for every broker in a high-availability cluster.
- **imq.cluster.clusterid** specifies the *cluster identifier*, which will be appended to the names of all database tables in the cluster's shared persistent store. The value of this property must be the same for all brokers in a given cluster, but must be unique for each cluster: no two running clusters may have the same cluster identifier.
- **imq.brokerid** is a *broker identifier* that must be unique for each broker in the cluster. Hence, this property must be set in each broker's instance configuration file rather than in a cluster configuration file.

High-Availability Clusters: JDBC Configuration Properties

The persistent data store for a high-availability cluster is maintained on a highly-available JDBC database.

The highly-available database may be Sun's MySQL Cluster Edition or High Availability Session Store (HADB), or it may be an open-source or third-party product such as Oracle Corporation's Real Application Clusters (RAC). As described in [“JDBC-Based Persistence Properties” on page 128](#), the `imq.persist.jdbc.dbVendor` broker property specifies the name of the database vendor, and all of the remaining JDBC-related properties are qualified with this vendor name.

The JDBC-related properties are discussed under [“JDBC-Based Persistence Properties” on page 128](#) and summarized in [Table 16–6](#). See the example configurations for MySQL and HADB in [Example 8–1](#) and [Example 8–2](#), respectively.

Note – In setting JDBC-related properties for a high-availability cluster, note the following vendor-specific issues:

- **MySQL Cluster Edition**

When using MySQL Cluster Edition as a highly-available database, you must specify the NDB Storage Engine rather than the InnoDB Storage Engine set by Message Queue by default. To specify the NDB Storage Engine, set the following broker property for all brokers in the cluster:

```
imq.persist.jdbc.mysql.tableoption=ENGINE=NDBCLUSTER
```

- **HADB**

When using HADB in a Sun Java Application Server environment, if the integration between Message Queue and Application Server is local (that is, there is a one-to-one relationship between Application Server instances and Message Queue brokers), the Application Server will automatically propagate HADB-related properties to each broker in the cluster. However, if the integration is remote (a single Application Server instance using an externally configured broker cluster), then it is your responsibility to configure the needed HADB properties explicitly.

High-Availability Clusters: Failure Detection Properties

The following configuration properties (listed in [Table 16–11](#)) specify the parameters for the exchange of heartbeat and status information within a high-availability cluster:

- **imq.cluster.heartbeat.hostname** specifies the host name (or IP address) for the heartbeat connection service.
- **imq.cluster.heartbeat.port** specifies the port number for the heartbeat connection service.
- **imq.cluster.heartbeat.interval** specifies the interval, in seconds, at which heartbeat packets are transmitted.
- **imq.cluster.heartbeat.threshold** specifies the number of missed heartbeat intervals after which a broker is considered suspect of failure.
- **imq.cluster.monitor.interval** specifies the interval, in seconds, at which to monitor a suspect broker's state information to determine whether it has failed.
- **imq.cluster.monitor.threshold** specifies the number of elapsed monitor intervals after which a suspect broker is considered to have failed.

Smaller values for these heartbeat and monitoring intervals will result in quicker reaction to broker failure, but at the cost of reduced performance and increased likelihood of false suspicions and erroneous failure detection.

Displaying a Cluster Configuration

To display information about a cluster’s configuration, use the Command utility’s `list bkr` subcommand:

```
imqcmd list bkr
```

This lists the current state of all brokers included in the cluster to which a given broker belongs. The broker states are described in the following table:

TABLE 10–1 Broker States

Broker State	Meaning
OPERATING	Broker is operating
TAKEOVER_STARTED	For high-availability clusters, broker has begun taking over persistent data store from another broker
TAKEOVER_COMPLETE	For high-availability clusters, broker has finished taking over persistent data store from another broker
TAKEOVER_FAILED	For high-availability clusters, attempted takeover has failed
QUIESCE_STARTED	Broker has begun quiescing
QUIESCE_COMPLETE	Broker has finished quiescing
SHUTDOWN_STARTED	Broker has begun shutting down
BROKER_DOWN	Broker is down
UNKNOWN	Broker state unknown

The results of the `imqcmd list bkr` command are shown in [Example 10–1](#) (for a conventional cluster) and [Example 10–2](#) (for a high-availability cluster).

EXAMPLE 10-1 Configuration Listing for a Conventional Cluster

Listing all the brokers in the cluster that the following broker is a member of:

```

-----
Host          Primary Port
-----
localhost     7676

Cluster Is Highly Available      False

-----
Address          State
-----
whippet:7676     OPERATING
greyhound:7676  OPERATING

```

EXAMPLE 10-2 Configuration Listing for a high-availability Cluster

Listing all the brokers in the cluster that the following broker is a member of:

```

-----
Host          Primary Port   Cluster Broker ID
-----
localhost     7676           brokerA

Cluster ID                myClusterID
Cluster Is Highly Available      True

-----
Broker ID      Address          State                Msgs in store   ID of broker
performing takeover      Time since last
status timestamp
-----
brokerA        localhost:7676  OPERATING            121              brokerA
brokerB        greyhound:7676 TAKEOVER_STARTED     52              3 hrs
brokerC        jpgserv:7676   SHUTDOWN_STARTED    12346            10 sec
brokerD        icdev:7676     TAKEOVER_COMPLETE    0                brokerA
brokerE        mrperf:7676    *unknown             12
brokerG        iclab1:7676    QUIESCING            4                2 sec
brokerH        iclab2:7676    QUIESCE_COMPLETE     8                5 sec

```

Managing Broker Clusters

The following sections describe how to perform various administrative management tasks for conventional and high-availability clusters, respectively.

Managing Conventional Clusters

The procedures in this section show how to perform the following tasks for a conventional cluster:

- [“Connecting Brokers into a Conventional Cluster” on page 178](#)
- [“Adding Brokers to a Conventional Cluster” on page 180](#)
- [“Removing Brokers From a Conventional Cluster” on page 181](#)
- [“Managing a Conventional Cluster’s Configuration Change Record” on page 182](#)

Connecting Brokers into a Conventional Cluster

There are two general methods of connecting brokers into a conventional cluster: from the command line (using the `-cluster` option) or by setting the `imq.cluster.brokerlist` property in the cluster configuration file. Whichever method you use, each broker that you start attempts to connect to the other brokers in the cluster every five seconds; the connection will succeed once the master broker is started up (if one is configured). If a broker in the cluster starts before the master broker, it will remain in a suspended state, rejecting client connections, until the master broker starts; the suspended broker then will automatically become fully functional. It is therefore a good idea to start the master broker first and then the others, after the master broker has completed its startup.

Note – Whichever method you use to connect brokers into a conventional cluster, you must make sure that no broker in the cluster is given an address that resolves to the network loopback IP address (127.0.0.1). Any broker configured with this address will be unable to connect to other brokers in the cluster.

In particular, some Linux installers automatically set the `localhost` entry to the network loopback address. On such systems, you must modify the system IP address so that all brokers in the cluster can be addressed properly: For each Linux system participating in the cluster, check the `/etc/hosts` file as part of cluster setup. If the system uses a static IP address, edit the `/etc/hosts` file to specify the correct address for `localhost`. If the address is registered with Domain Name Service (DNS), edit the file `/etc/nsswitch.conf` to change the order of the entries so that DNS lookup is performed before consulting the local `hosts` file. The line in `/etc/nsswitch.conf` should read as follows:

```
hosts: dns files
```

Note – On a multi-homed computer, in which there is more than one network interface card, be sure to explicitly set the network interface to be used by the broker for client connection services (`imq.hostname`) *and* for the cluster connection service (`imq.cluster.hostname`). If `imq.cluster.hostname` is not set, then connections between brokers might not succeed and as a result, the cluster will not be established.

Note – If you are clustering a Message Queue version 4.1 or later broker (for example, 4.2) together with those from earlier versions than Message Queue 4.1, you must set the value of the 4.1 or later broker's `imq.autocreate.queue.maxNumActiveConsumers` property to 1. Otherwise the brokers will not be able to establish a cluster connection.

▼ To Connect Brokers from the Command Line

- 1 **If you are using a master broker, start it with the `imqbrokerd` command, using the `-cluster` option to specify the complete list of brokers to be included in the cluster.**

For example, the following command starts the broker as part of a cluster consisting of the brokers running at the default port (7676) on `host1`, at port 5000 on `host2`, and at port 9876 on the default host (`localhost`):

```
imqbrokerd -cluster host1,host2:5000,:9876
```

- 2 **Once the master broker (if any) is running, start each of the other brokers in the cluster with the `imqbrokerd` command, using the same list of brokers with the `-cluster` option that you used for the master broker.**

The value specified for the `-cluster` option must be the same for all brokers in the cluster.

▼ To Connect Brokers Using a Cluster Configuration File

An alternative method, better suited for production systems, is to use a cluster configuration file to specify the composition of the cluster:

- 1 **Create a cluster configuration file that uses the `imq.cluster.brokerlist` property to specify the list of brokers to be connected.**

If you are using a master broker, identify it with the `imq.cluster.masterbroker` property in the configuration file.

- 2 **For each broker in the cluster, set the `imq.cluster.url` property in the broker's instance configuration file to point to the cluster configuration file.**

- 3 **Use the `imqbrokerd` command to start each broker.**

If there is a master broker, start it first, then the others after it has completed its startup.

▼ To Establish Secure Connections Between Brokers

If you want secure, encrypted message delivery between brokers in a cluster, configure the cluster connection service to use an SSL-based transport protocol:

- 1 For each broker in the cluster, set up SSL-based connection services, as described in [“Message Encryption” on page 159](#).
- 2 Set each broker’s `imq.cluster.transport` property to `ssl`, either in the cluster configuration file or individually for each broker.

Adding Brokers to a Conventional Cluster

The procedure for adding a new broker to a conventional cluster depends on whether the cluster uses a cluster configuration file.

▼ To Add a New Broker to a Conventional Cluster Using a Cluster Configuration File

- 1 Add the new broker to the `imq.cluster.brokerlist` property in the cluster configuration file.
- 2 Issue the following command to any broker in the cluster:

```
imqcmd reload cls
```

This forces each broker to reload the cluster configuration, ensuring that all persistent information for brokers in the cluster is up to date. Note that it is not necessary to issue this command to every broker in the cluster; executing it for any one broker will cause all of them to reload the cluster configuration.

- 3 *(Optional)* Set the value of the `imq.cluster.url` property in the new broker’s instance configuration file (`config.properties`) to point to the cluster configuration file.
- 4 Start the new broker.

If you did not perform step 3, use the `-D` option on the `imqbrokerd` command line to set the value of `imq.cluster.url` to the location of the cluster configuration file.

▼ To Add a New Broker to a Conventional Cluster Without a Cluster Configuration File

- 1 *(Optional)* Set the values of the following properties in the new broker’s instance configuration file (`config.properties`):

```
imq.cluster.brokerlist
imq.cluster.masterbroker (if necessary)
```

`imq.cluster.transport` (if you are using a secure cluster connection service)

When the newly added broker starts, it connects and exchanges data with all the other brokers in the `imq.cluster.brokerlist` value.

2 Modify the `imq.cluster.brokerlist` property of other brokers in the cluster to include the new broker.

This step is not strictly necessary to add a broker to a functioning cluster. However, should any broker need to be restarted, its `imq.cluster.brokerlist` value must include all other brokers in the cluster, including the newly added broker.

3 Start the new broker.

If you did not perform step 1, use the `-D` option on the `imqbrokerd` command line to set the property values listed there.

Removing Brokers From a Conventional Cluster

The method you use to remove a broker from a conventional cluster depends on whether you originally created the cluster from the command line or by means of a central cluster configuration file.

▼ To Remove a Broker From a Conventional Cluster Using a Cluster Configuration File

If you originally created a cluster by specifying its member brokers with the `imq.cluster.brokerlist` property in a central cluster configuration file, it isn't necessary to stop the brokers in order to remove one of them. Instead, you can simply edit the configuration file to exclude the broker you want to remove, force the remaining cluster members to reload the cluster configuration, and reconfigure the excluded broker so that it no longer points to the same cluster configuration file:

1 Edit the cluster configuration file to remove the excluded broker from the list specified for the `imq.cluster.brokerlist` property.

2 Issue the following command to each broker remaining in the cluster:

```
imqcmd reload cls
```

This forces the brokers to reload the cluster configuration.

3 Stop the broker you're removing from the cluster.

4 Edit that broker's instance configuration file (`config.properties`), removing or specifying a different value for its `imq.cluster.url` property.

▼ To Remove a Broker From a Conventional Cluster Using the Command Line

If you used the `imqbrokerd` command from the command line to connect the brokers into a cluster, you must stop each of the brokers and then restart them, specifying the new set of cluster members on the command line:

- 1 **Stop each broker in the cluster, using the `imqcmd` command.**
- 2 **Restart the brokers that will remain in the cluster, using the `imqbrokerd` command's `-cluster` option to specify only those remaining brokers.**

For example, suppose you originally created a cluster consisting of brokers *A*, *B*, and *C* by starting each of the three with the command

```
imqbrokerd -cluster A,B,C
```

To remove broker *A* from the cluster, restart brokers *B* and *C* with the command

```
imqbrokerd -cluster B,C
```

Managing a Conventional Cluster's Configuration Change Record

As noted earlier, a conventional cluster can optionally have one master broker, which maintains a configuration change record to keep track of any changes in the cluster's persistent state. The master broker is identified by the `imq.cluster.masterbroker` configuration property, either in the cluster configuration file or in the instance configuration files of the individual brokers.

Because of the important information that the configuration change record contains, it is important to back it up regularly so that it can be restored in case of failure. Although restoring from a backup will lose any changes in the cluster's persistent state that have occurred since the backup was made, frequent backups can minimize this potential loss of information. The backup and restore operations also have the positive effect of compressing and optimizing the change history contained in the configuration change record, which can grow significantly over time.

▼ To Back Up the Configuration Change Record

- **Use the `-backup` option of the `imqbrokerd` command, specifying the name of the backup file.**

For example:

```
imqbrokerd -backup mybackuplog
```

▼ To Restore the Configuration Change Record

- 1 Shut down all brokers in the cluster.
- 2 Restore the master broker's configuration change record from the backup file.
The command is

```
imqbrokerd -restore mybackuplog
```
- 3 If you assign a new name or port number to the master broker, update the `imq.cluster.brokerlist` and `imq.cluster.masterbroker` properties accordingly in the cluster configuration file.
- 4 Restart all brokers in the cluster.

Managing High-Availability Clusters

This section presents step-by-step procedures for performing a variety of administrative tasks for a high-availability cluster:

- “Connecting Brokers into a High-Availability Cluster” on page 183
- “Adding and Removing Brokers in a High-Availability Cluster” on page 185
- “Preventing or Forcing Broker Failover” on page 186
- “Converting a Standalone Data Store to a Shared Data Store” on page 187
- “Backing up a Shared Data Store” on page 187

Connecting Brokers into a High-Availability Cluster

Because high-availability clusters are self-configuring, there is no need to explicitly specify the list of brokers to be included in the cluster. Instead, all that is needed is to set each broker's configuration properties appropriately and then start the broker; as long as its properties are set properly, it will automatically be incorporated into the cluster. “[High-Availability Cluster Properties](#)” on page 173 describes the required properties, which include vendor-specific JDBC database properties.

Note – In addition to creating a high-availability cluster as described in this section, you must also configure clients to successfully reconnect to a failover broker in the event of broker or connection failure. You do this by setting the `imqReconnectAttempts` connection factory attribute to a value of -1.

The property values needed for brokers in a high-availability cluster can be set separately in each broker's instance configuration file, or they can be specified in a cluster configuration file that all the brokers reference. The procedures are as follows:

▼ To Connect Brokers Using a Cluster Configuration File

An alternative method, better suited for production systems, is to use a cluster configuration file to specify the composition of the cluster:

- 1 **Create a cluster configuration file specifying the cluster's high-availability-related configuration properties.**

[“High-Availability Cluster Properties” on page 173](#) shows the required property values.

However, do *not* include the `imq.brokerid` property in the cluster configuration file; this must be specified separately for each individual broker in the cluster.

- 2 **Specify any additional, vendor-specific properties that might be needed.**

The vendor-specific properties required for MySQL and HADB are shown in [Example 8–1](#) and [Example 8–2](#), respectively.

- 3 **For each broker in the cluster:**

- a. **Start the broker with the `imqbrokerd` command.**

The first time a broker instance is run, an instance configuration file (`config.properties`) is automatically created.

- b. **Shut down the broker.**

Use the `imqcmd shutdown bkr` command.

- c. **Edit the instance configuration file to specify the location of the cluster configuration file.**

In the broker's instance configuration file, set the `imq.cluster.url` property to point to the location of the cluster configuration file you created in step 1.

- d. **Specify the broker identifier.**

Set the `imq.brokerid` property in the instance configuration file to the broker's unique broker identifier. This value must be different for each broker.

- 4 **Place a copy of, or a symbolic link to, your JDBC driver's `.jar` file in the Message Queue external resource files directory, depending on your platform (see [Appendix A, “Platform-Specific Locations of Message Queue Data”](#)):**

Solaris: `/usr/share/lib/imq/ext`

Linux: `/opt/sun/mq/share/lib/ext`

Windows: `IMQ_VARHOME\lib\ext`

- 5 **Create the database schema needed for Message Queue persistence.**

Use the `imqdbmgr create tbl` command; see [“Database Manager Utility” on page 294](#).

6 Restart each broker with the `imqbrokerd` command.

The brokers will automatically register themselves into the cluster on startup.

▼ To Connect Brokers Using Instance Configuration Files

1 For each broker in the cluster:

a. Start the broker with the `imqbrokerd` command.

The first time a broker instance is run, an instance configuration file (`config.properties`) is automatically created.

b. Shut down the broker.

Use the `imqcmd shutdown bkr` command.

c. Edit the instance configuration file to specify the broker's high-availability-related configuration properties.

[“High-Availability Cluster Properties” on page 173](#) shows the required property values.

d. Specify any additional, vendor-specific properties that might be needed.

The vendor-specific properties required for MySQL and HADB are shown in [Example 8-1](#) and [Example 8-2](#), respectively.

2 Place a copy of, or a symbolic link to, your JDBC driver's `.jar` file in the Message Queue external resource files directory, depending on your platform (see [Appendix A, “Platform-Specific Locations of Message Queue Data”](#)):

Solaris: `/usr/share/lib/imq/ext`

Linux: `/opt/sun/mq/share/lib/ext`

Windows: `IMQ_VARHOME\lib\ext`

3 Create the database schema needed for Message Queue persistence.

Use the `imqdbmgr create tbl` command; see [“Database Manager Utility” on page 294](#).

4 Restart each broker with the `imqbrokerd` command.

The brokers will automatically register themselves into the cluster on startup.

Adding and Removing Brokers in a High-Availability Cluster

Because high-availability clusters are self-configuring, the procedures for adding and removing brokers are simpler than for a conventional cluster.

▼ To Add a New Broker to a High-availability Cluster

- 1 **Set the new broker's high-availability-related properties, as described in the preceding section.**

You can do this either by specifying the individual properties in the broker's instance configuration file (`config.properties`) or, if there is a cluster configuration file, by setting the broker's `imq.cluster.url` property to point to it.

- 2 **Start the new broker with the `imqbrokerd` command.**

The broker will automatically register itself into the cluster on startup.

▼ To Remove a Broker from a high-availability Cluster

- 1 **Make sure the broker is not running.**

If necessary, use the command

```
imqcmd shutdown bkr
```

to shut down the broker.

- 2 **Remove the broker from the cluster with the command**

```
imqdbmgr remove bkr
```

This command deletes all database tables for the corresponding broker.

Preventing or Forcing Broker Failover

Although the takeover of a failed broker's persistent data by a failover broker in a high-availability cluster is normally automatic, there may be times when you want to prevent such failover from occurring. To suppress automatic failover when shutting down a broker, use the `-nofailover` option to the `imqcmd shutdown bkr` subcommand:

```
imqcmd shutdown bkr -nofailover -b hostName:portNumber
```

where *hostName* and *portNumber* are the host name and port number of the broker to be shut down.

Conversely, you may sometimes need to force a broker failover to occur manually. (This might be necessary, for instance, if a failover broker were to itself fail before completing the takeover process.) In such cases, you can initiate a failover manually from the command line: first shut down the broker to be taken over with the `-nofailover` option, as shown above, then issue the command

```
imqcmd takeover bkr -n brokerID
```

where *brokerID* is the broker identifier of the broker to be taken over. If the specified broker appears to be running, the Command utility will display a confirmation message:

The broker associated with *brokerID* last accessed the database # seconds ago.
Do you want to take over for this broker?

You can suppress this message, and force the takeover to occur unconditionally, by using the `-f` option to the `imqcmd takeover bkr` command:

```
imqcmd takeover bkr -f -n brokerID
```

Note – The `imqcmd takeover bkr` subcommand is intended only for use in failed-takeover situations. You should use it only as a last resort, and not as a general way of forcibly taking over a running broker.

Converting a Standalone Data Store to a Shared Data Store

When converting to high-availability operation using an HADB database, you can use the Message Queue Database Manager utility (`imqdbmgr`) subcommand

```
imqdbmgr upgrade hystore
```

to convert an existing standalone HADB database to a shared HADB database. You can use this command in the following cases:

- Migrating from a Message Queue 4.0 or 4.1 standalone HADB database to a Message Queue 4.2 shared HADB database. In this case, the broker, upon startup, will automatically upgrade the database schema to Message Queue 4.2. You can then run the `imqdbmgr upgrade hystore` command to convert the data store to shared use.
- Moving from a standalone Message Queue 4.21 HADB database to a shared HADB database. In this case, you just need to run the `imqdbmgr upgrade hystore` command to convert the data store to shared use.

Because this command only supports conversion of HADB stores, it cannot be used to convert file-based stores or other JDBC-based stores to a shared HADB store. If you were previously running a 3.x version of Message Queue, you must create an HADB store and then manually migrate your data to that store in order to use the high availability feature.

Backing up a Shared Data Store

For durability and reliability, it is a good idea to back up a high-availability cluster's shared data store periodically to backup files. This creates a snapshot of the data store that you can then use to restore the data in case of catastrophic failure. The command for backing up the data store is

```
imqdbmgr backup -dir backupDir
```

where *backupDir* is the path to the directory in which to place the backup files. To restore the data store from these files, use the command

```
imqdbmgr restore -restore backupDir
```


Managing Administered Objects

Administered objects encapsulate provider-specific configuration and naming information, enabling the development of client applications that are portable from one JMS provider to another. A Message Queue administrator typically creates administered objects for client applications to use in obtaining broker connections for sending and receiving messages.

This chapter tells how to use the Object Manager utility (`imqobjmgr`) to create and manage administered objects. It contains the following sections:

- [“Object Stores” on page 189](#)
- [“Administered Object Attributes” on page 192](#)
- [“Using the Object Manager Utility” on page 199](#)

Object Stores

Administered objects are placed in a readily available *object store* where they can be accessed by client applications by means of the Java Naming and Directory Interface (JNDI). There are two types of object store you can use: a standard Lightweight Directory Access Protocol (LDAP) directory server or a directory in the local file system.

LDAP Server Object Stores

An LDAP server is the recommended object store for production messaging systems. LDAP servers are designed for use in distributed systems and provide security features that are useful in production environments.

LDAP implementations are available from a number of vendors. To manage an object store on an LDAP server with Message Queue administration tools, you may first need to configure the server to store Java objects and perform JNDI lookups; see the documentation provided with your LDAP implementation for details.

To use an LDAP server as your object store, you must specify the attributes shown in Table 11–1. These attributes fall into the following categories:

- **Initial context.** The `java.naming.factory.initial` attribute specifies the initial context for JNDI lookups on the server. The value of this attribute is fixed for a given LDAP object store.
- **Location.** The `java.naming.provider.url` attribute specifies the URL and directory path for the LDAP server. You must verify that the specified directory path exists.
- **Security.** The `java.naming.security.principal`, `java.naming.security.credentials`, and `java.naming.security.authentication` attributes govern the authentication of callers attempting to access the object store. The exact format and values of these attributes depend on the LDAP service provider; see the documentation provided with your LDAP implementation for details and to determine whether security information is required on all operations or only on those that change the stored data.

TABLE 11–1 LDAP Object Store Attributes

Attribute	Description
<code>java.naming.factory.initial</code>	Initial context for JNDI lookup Example: <code>com.sun.jndi.ldap.LdapCtxFactory</code>
<code>java.naming.provider.url</code>	Server URL and directory path Example: <code>ldap://myD.com:389/ou=mq1,o=App</code> where administered objects are stored in the directory <code>/App/mq1</code> .
<code>java.naming.security.principal</code>	Identity of the principal for authenticating callers The format of this attribute depends on the authentication scheme: for example, <code>uid=homerSimpson,ou=People,o=mq</code> If this attribute is unspecified, the behavior is determined by the LDAP service provider.
<code>java.naming.security.credentials</code>	Credentials of the authentication principal The value of this attribute depends on the authentication scheme: for example, it might be a hashed password, a clear-text password, a key, or a certificate. If this property is unspecified, the behavior is determined by the LDAP service provider.

TABLE 11-1 LDAP Object Store Attributes (Continued)

Attribute	Description
<code>java.naming.security.authentication</code>	<p>Security level for authentication:</p> <ul style="list-style-type: none"> <code>none</code>: No security <code>simple</code>: Simple security <code>strong</code>: Strong security <p>For example, if you specify <code>simple</code>, you will be prompted for any missing principal or credential values. This will allow you a more secure way of providing identifying information.</p> <p>If this property is unspecified, the behavior is determined by the LDAP service provider.</p>

File-System Object Stores

Message Queue also supports the use of a directory in the local file system as an object store for administered objects. While this approach is not recommended for production systems, it has the advantage of being very easy to use in development environments. Note, however, that for a directory to be used as a centralized object store for clients deployed across multiple computer nodes, all of those clients must have access to the directory. In addition, any user with access to the directory can use Message Queue administration tools to create and manage administered objects.

To use a file-system directory as your object store, you must specify the attributes shown in [Table 11-2](#). These attributes have the same general meanings described above for LDAP object stores; in particular, the `java.naming.provider.url` attribute specifies the directory path of the directory holding the object store. This directory must exist and have the proper access permissions for the user of Message Queue administration tools as well as the users of the client applications that will access the store.

TABLE 11-2 File-system Object Store Attributes

Attribute	Description
<code>java.naming.factory.initial</code>	<p>Initial context for JNDI lookup</p> <p>Example:</p> <pre>com.sun.jndi.fscontext.ReffFSContextFactory</pre>
<code>java.naming.provider.url</code>	<p>Directory path</p> <p>Example:</p> <pre>file:///C:/myapp/mqobj.s</pre>

Administered Object Attributes

Message Queue administered objects are of two basic kinds:

- *Connection factories* are used by client applications to create connections to brokers.
- *Destinations* represent locations on a broker with which client applications can exchange (send and retrieve) messages.

Each type of administered object has certain attributes that determine the object's properties and behavior. This section describes how to use the Object Manager command line utility (`imqobjmgr`) to set these attributes; you can also set them with the GUI Administration Console, as described in [“Working With Administered Objects” on page 56](#).

Connection Factory Attributes

Client applications use *connection factory* administered objects to create connections with which to exchange messages with a broker. A connection factory's attributes define the properties of all connections it creates. Once a connection has been created, its properties cannot be changed; thus the only way to configure a connection's properties is by setting the attributes of the connection factory used to create it.

Message Queue defines two classes of connection factory objects:

- `ConnectionFactory` objects support normal messaging and nondistributed transactions.
- `XAConnectionFactory` objects support distributed transactions.

Both classes share the same configuration attributes, which you can use to optimize resources, performance, and message throughput. These attributes are listed and described in detail in [Chapter 18, “Administered Object Attribute Reference,”](#) and are discussed in the following sections below:

- [“Connection Handling” on page 192](#)
- [“Client Identification” on page 195](#)
- [“Reliability And Flow Control” on page 197](#)
- [“Queue Browser and Server Sessions” on page 198](#)
- [“Standard Message Properties” on page 198](#)
- [“Message Header Overrides” on page 198](#)

Connection Handling

Connection handling attributes specify the broker address to which to connect and, if required, how to detect connection failure and attempt reconnection. They are summarized in [Table 18–1](#).

Broker Address List

The most important connection handling attribute is `imqAddressList`, which specifies the broker or brokers to which to establish a connection. The value of this attribute is a string containing a broker address or (in the case of a broker cluster) multiple addresses separated by commas. Broker addresses can use a variety of addressing schemes, depending on the connection service to be used (see “[Configuring Connection Services](#)” on page 93) and the method of establishing a connection:

- `mq` uses the broker’s Port Mapper to assign a port dynamically for either the `jms` or `ssljms` connection service.
- `mqtcp` bypasses the Port Mapper and connects directly to a specified port, using the `jms` connection service.
- `mqssl` makes a Secure Socket Layer (SSL) connection to a specified port, using the `ssljms` connection service.
- `http` makes a Hypertext Transport Protocol (HTTP) connection to a Message Queue tunnel servlet at a specified URL, using the `httpjms` connection service.
- `https` makes a Secure Hypertext Transport Protocol (HTTPS) connection to a Message Queue tunnel servlet at a specified URL, using the `httpsjms` connection service.

These addressing schemes are summarized in [Table 18–2](#).

The general format for each broker address is

scheme://*address*

where *scheme* is one of the addressing schemes listed above and *address* denotes the broker address itself. The exact syntax for specifying the address varies depending on the addressing scheme, as shown in the “Description” column of [Table 18–2](#). [Table 18–3](#) shows examples of the various address formats.

In a multiple-broker cluster environment, the address list can contain more than one broker address. If the first connection attempt fails, the Message Queue client runtime will attempt to connect to another address in the list, and so on until the list is exhausted. Two additional connection factory attributes control the way this is done:

- `imqAddressListBehavior` specifies the order in which to try the specified addresses. If this attribute is set to the string `PRIORITY`, addresses will be tried in the order in which they appear in the address list. If the attribute value is `RANDOM`, the addresses will instead be tried in random order; this is useful, for instance, when many Message Queue clients are sharing the same connection factory object, to prevent them from all attempting to connect to the same broker address.
- `imqAddressListIterations` specifies how many times to cycle through the list before giving up and reporting failure. A value of `–1` denotes an unlimited number of iterations: the client runtime will keep trying until it succeeds in establishing a connection or until the end of time, whichever occurs first.

Note – Because high-availability clusters are self-configuring (see [“Cluster Configuration Properties” on page 172](#) and [“Connecting Brokers into a High-Availability Cluster” on page 183](#)), their membership can change over time as brokers enter and leave the cluster. In this type of cluster, the value of each member broker’s `imqAddressList` attribute is updated dynamically so that it always reflects the cluster’s current membership.

Automatic Reconnection

By setting certain connection factory attributes, you can configure a client to reconnect automatically to a broker in the event of a failed connection. For standalone brokers or those belonging to a conventional broker cluster (see [“Conventional Clusters” in *Sun Java System Message Queue 4.2 Technical Overview*](#)), you enable this behavior by setting the connection factory’s `imqReconnectEnabled` attribute to `true`. The `imqReconnectAttempts` attribute controls the number of reconnection attempts to a given broker address; `imqReconnectInterval` specifies the interval, in milliseconds, to wait between attempts.

If the broker is part of a conventional cluster, the failed connection can be restored not only on the original broker, but also on a different one in the cluster. If reconnection to the original broker fails, the client runtime will try the other addresses in the connection factory’s broker address list (`imqAddressList`). The `imqAddressListBehavior` and `imqAddressListIterations` attributes control the order in which addresses are tried and the number of iterations through the list, as described in the preceding section. Each address is tried repeatedly at intervals of `imqReconnectInterval` milliseconds, up to the maximum number of attempts specified by `imqReconnectAttempts`.

Note, however, that in a conventional cluster, such automatic reconnection only provides connection failover and not data failover: persistent messages and other state information held by a failed or disconnected broker can be lost when the client is reconnected to a different broker instance. While attempting to reestablish a connection, Message Queue does maintain objects (such as sessions, message consumers, and message producers) provided by the client runtime. Temporary destinations are also maintained for a time when a connection fails, because clients might reconnect and access them again; after giving clients time to reconnect and use these destinations, the broker will delete them. In circumstances where the client-side state cannot be fully restored on the broker on reconnection (for instance, when using transacted sessions, which exist only for the duration of a connection), automatic reconnection will not take place and the connection’s exception handler will be called instead. It is then up to the client application to catch the exception, reconnect, and restore state.

By contrast, in a high-availability cluster (see [“High-Availability Clusters” in *Sun Java System Message Queue 4.2 Technical Overview*](#)), another broker can take over a failed broker’s persistent state and proceed to deliver its pending messages without interruption of service. In this type of cluster, automatic reconnection is always enabled. The connection factory’s `imqReconnectEnabled`, `imqAddressList`, and `imqAddressListIterations` attributes are ignored. The client runtime is automatically redirected to the failover broker. Because there

might be a short time lag during which the failover broker takes over from the failed broker, the `imqReconnectAttempts` connection factory attribute should be set to a value of -1 (client runtime continues connect attempts until successful).

Automatic reconnection supports all client acknowledgment modes for message consumption. Once a connection has been reestablished, the broker will redeliver all unacknowledged messages it had previously delivered, marking them with a `Redeliver` flag. Client applications can use this flag to determine whether a message has already been consumed but not yet acknowledged. (In the case of nondurable subscribers, however, the broker does not hold messages once their connections have been closed. Thus any messages produced for such subscribers while the connection is down cannot be delivered after reconnection and will be lost.) Message production is blocked while automatic reconnection is in progress; message producers cannot send messages to the broker until after the connection has been reestablished.

Periodic Testing (Pinging) of Connections

The Message Queue client runtime can be configured to periodically test, or “ping,” a connection, allowing connection failures to be detected preemptively before an attempted message transmission fails. Such testing is particularly important for client applications that only consume messages and do not produce them, since such applications cannot otherwise detect when a connection has failed. Clients that produce messages only infrequently can also benefit from this feature.

The connection factory attribute `imqPingInterval` specifies the frequency, in seconds, with which to ping a connection. By default, this interval is set to 30 seconds; a value of -1 disables the ping operation.

The response to an unsuccessful ping varies from one operating-system platform to another. On some platforms, an exception is immediately thrown to the client application’s exception listener. (If the client does not have an exception listener, its next attempt to use the connection will fail.) Other platforms may continue trying to establish a connection to the broker, buffering successive pings until one succeeds or the buffer overflows.

Client Identification

The connection factory attributes listed in [Table 18–4](#) support client authentication and the setting of client identifiers for durable subscribers.

Client Authentication

All attempts to connect to a broker must be authenticated by user name and password against a user repository maintained by the message service. The connection factory attributes `imqDefaultUsername` and `imqDefaultPassword` specify a default user name and password to be used if the client does not supply them explicitly when creating a connection.

As a convenience for developers who do not wish to bother populating a user repository during application development and testing, Message Queue provides a guest user account with user name and password both equal to `guest`. This is also the default value for the `imqDefaultUsername` and `imqDefaultPassword` attributes, so that if they are not specified explicitly, clients can always obtain a connection under the guest account. In a production environment, access to broker connections should be restricted to users who are explicitly registered in the user repository.

Client Identifier

The *Java Message Service Specification* requires that a connection provide a unique *client identifier* whenever the broker must maintain a persistent state on behalf of a client. Message Queue uses such client identifiers to keep track of durable subscribers to a topic destination. When a durable subscriber becomes inactive, the broker retains all incoming messages for the topic and delivers them when the subscriber becomes active again. The broker identifies the subscriber by means of its client identifier.

While it is possible for a client application to set its own client identifier programmatically using the connection object's `setClientID` method, this makes it difficult to coordinate client identifiers to ensure that each is unique. It is generally better to have Message Queue automatically assign a unique identifier when creating a connection on behalf of a client. This can be done by setting the connection factory's `imqConfiguredClientID` attribute to a value of the form

`${u}factoryID`

The characters `${u}` must be the first four characters of the attribute value. (Any character other than `u` between the braces will cause an exception to be thrown on connection creation; in any other position, these characters have no special meaning and will be treated as plain text.) The value for *factoryID* is a character string uniquely associated with this connection factory object.

When creating a connection for a particular client, Message Queue will construct a client identifier by replacing the characters `${u}` with `${u:userName}`, where *userName* is the user name authenticated for the connection. This ensures that connections created by a given connection factory, although identical in all other respects, will each have their own unique client identifier. For example, if the user name is `Calvin` and the string specified for the connection factory's `imqConfiguredClientID` attribute is `${u}Hobbes`, the client identifier assigned will be `${u:Calvin}Hobbes`.

Note – This scheme will not work if two clients both attempt to obtain connections using the default user name `guest`, since each would have a client identifier with the same `#{u}` component. In this case, only the first client to request a connection will get one; the second client's connection attempt will fail, because Message Queue cannot create two connections with the same client identifier.

Even if you specify a client identifier with `imqConfiguredClientID`, client applications can override this setting with the connection method `setClientID`. You can prevent this by setting the connection factory's `imqDisableSetClientID` attribute to `true`. Note that for an application that uses durable subscribers, the client identifier *must* be set one way or the other: either administratively with `imqConfiguredClientID` or programmatically with `setClientID`.

Reliability And Flow Control

Because “payload” messages sent and received by clients and control messages (such as broker acknowledgments) used by Message Queue itself pass over the same client-broker connection, excessive levels of payload traffic can interfere with the delivery of control messages. To help alleviate this problem, the connection factory attributes listed in [Table 18–5](#) allow you to manage the relative flow of the two types of message. These attributes fall into four categories:

- **Acknowledgment timeout** specifies the maximum time (`imqAckTimeout`) to wait for a broker acknowledgment before throwing an exception.
- **Connection flow metering** limits the transmission of payload messages to batches of a specified size (`imqConnectionFlowCount`), ensuring periodic opportunities to deliver any accumulated control messages.
- **Connection flow control** limits the number of payload messages (`imqConnectionFlowLimit`) that can be held pending on a connection, waiting to be consumed. When the limit is reached, delivery of payload messages to the connection is suspended until the number of messages awaiting consumption falls below the limit. Use of this feature is controlled by a boolean flag (`imqConnectionFlowLimitEnabled`).
- **Consumer flow control** limits the number of payload messages (`imqConsumerFlowLimit`) that can be held pending for any single consumer, waiting to be consumed. (This limit can also be specified as a property of a specific queue destination, `consumerFlowLimit`.) When the limit is reached, delivery of payload messages to the consumer is suspended until the number of messages awaiting consumption, as a percentage of `imqConsumerFlowLimit`, falls below the limit specified by the `imqConsumerFlowThreshold` attribute. This helps improve load balancing among multiple consumers by preventing any one consumer from starving others on the same connection.

The use of any of these flow control techniques entails a trade-off between reliability and throughput; see “[Client Runtime Message Flow Adjustments](#)” on [page 246](#) for further discussion.

Queue Browser and Server Sessions

[Table 18–6](#) lists connection factory attributes affecting client queue browsing and server sessions. The `imqQueueBrowserMaxMessagesPerRetrieve` attribute specifies the maximum number of messages to retrieve at one time when browsing the contents of a queue destination; `imqQueueBrowserRetrieveTimeout` gives the maximum waiting time for retrieving them. (Note that `imqQueueBrowserMaxMessagesPerRetrieve` does not affect the total number of messages browsed, only the way they are batched for delivery to the client runtime: fewer but larger batches or more but smaller ones. Changing the attribute's value may affect performance, but will not affect the total amount of data retrieved; the client application will always receive all messages in the queue.) The boolean attribute `imqLoadMaxToServerSession` governs the behavior of connection consumers in an application server session: if the value of this attribute is `true`, the client will load up to the maximum number of messages into a server session; if `false`, it will load only a single message at a time.

Standard Message Properties

The *Java Message Service Specification* defines certain standard message properties, which JMS providers (such as Message Queue) may optionally choose to support. By convention, the names of all such standard properties begin with the letters `JMSX`. The connection factory attributes listed in [Table 18–7](#) control whether the Message Queue client runtime sets certain of these standard properties. For produced messages, these include the following:

<code>JMSXUserID</code>	Identity of the user sending the message
<code>JMSXAppID</code>	Identity of the application sending the message
<code>JMSXProducerTXID</code>	Transaction identifier of the transaction within which the message was produced

For consumed messages, they include

<code>JMSXConsumerTXID</code>	Transaction identifier of the transaction within which the message was consumed
<code>JMSXRcvTimestamp</code>	Time the message was delivered to the consumer

Message Header Overrides

You can use the connection factory attributes listed in [Table 18–8](#) to override the values set by a client for certain JMS message header fields. The settings you specify will be used for all messages produced by connections obtained from that connection factory. Header fields that you can override in this way are

<code>JMSDeliveryMode</code>	Delivery mode (persistent or nonpersistent)
<code>JMSExpiration</code>	Expiration time
<code>JMSPriority</code>	Priority level

There are two attributes for each of these fields: one boolean, to control whether the field can be overridden, and another to specify its value. For instance, the attributes for setting the priority level are `imqOverrideJMSPriority` and `imqJMSPriority`. There is also an additional attribute, `imqOverrideJMSHeadersToTemporaryDestinations`, that controls whether override values apply to temporary destinations.

Note – Because overriding message headers may interfere with the needs of specific applications, these attributes should only be used in consultation with an application’s designers or users.

Destination Attributes

The *destination* administered object that identifies a physical queue or topic destination has only two attributes, listed in [Table 18–9](#). The important one is `imqDestinationName`, which gives the name of the physical destination that this administered object represents; this is the name that was specified with the `-n` option to the `imqcmd create dst` command that created the physical destination. (Note that there is not necessarily a one-to-one relationship between destination administered objects and the physical destinations they represent: a single physical destination can be referenced by more than one administered object, or by none at all.) There is also an optional descriptive string, `imqDestinationDescription`, which you can use to help identify the destination object and distinguish it from others you may have created.

Using the Object Manager Utility

The Message Queue Object Manager utility (`imqobjmgr`) allows you to create and manage administered objects. The `imqobjmgr` command provides the following subcommands for performing various operations on administered objects:

<code>add</code>	Add an administered object to an object store
<code>delete</code>	Delete an administered object from an object store
<code>list</code>	List existing administered objects in an object store
<code>query</code>	Display information about an administered object
<code>update</code>	Modify the attributes of an administered object

See “Object Manager Utility” on page 292 for reference information about the syntax, subcommands, and options of the `imqobjmgr` command.

Most Object Manager operations require you to specify the following information as options to the `imqobjmgr` command:

- The **JNDI lookup name** (`-l`) of the administered object

This is the logical name by which client applications can look up the administered object in the object store, using the Java Naming and Directory Interface.

- The **attributes of the JNDI object store** (- j)

See “[Object Stores](#)” on page 189 for information on the possible attributes and their values.

- The **type** (- t) of the administered object

Possible types include the following:

q	Queue destination
t	Topic destination
cf	Connection factory
qf	Queue connection factory
tf	Topic connection factory
xcf	Connection factory for distributed transactions
xqf	Queue connection factory for distributed transactions
xtf	Topic connection factory for distributed transactions

- The **attributes** (- o) of the administered object

See “[Administered Object Attributes](#)” on page 192 for information on the possible attributes and their values.

Adding Administered Objects

The `imqobjmgr` command’s `add` subcommand adds administered objects for connection factories and topic or queue destinations to the object store. Administered objects stored in an LDAP object store must have lookup names beginning with the prefix `cn=`; lookup names in a file-system object store need not begin with any particular prefix, but must not include the slash character (`/`).

Note – The Object Manager lists and displays only Message Queue administered objects. If an object store contains a non–Message Queue object with the same lookup name as an administered object that you wish to add, you will receive an error when you attempt the add operation.

Adding a Connection Factory

To enable client applications to create broker connections, add a connection factory administered object for the type of connection to be created: a queue connection factory or a topic connection factory, as the case may be. [Example 11–1](#) shows a command to add a queue connection factory (administered object type `qf`) to an LDAP object store. The object has lookup name `cn=myQCF` and connects to a broker running on host `myHost` at port number 7272, using the `jms` connection service.

EXAMPLE 11–1 Adding a Connection Factory

```
imqobjmgr add
-l "cn=myQCF"
-j "java.naming.factory.initial=com.sun.jndi.ldap.LdapCtxFactory"
-j "java.naming.provider.url=ldap://mydomain.com:389/o=imq"
-j "java.naming.security.principal=uid=homerSimpson,ou=People,o=imq"
-j "java.naming.security.credentials=doh"
-j "java.naming.security.authentication=simple"
-t qf
-o "imqAddressList=mq://myHost:7272/jms"
```

Adding a Destination

When creating an administered object representing a destination, it is good practice to create the physical destination first, before adding the administered object to the object store. Use the Command utility (`imqcmd`) to create the physical destination, as described in [“Creating and Destroying Physical Destinations” on page 107](#).

The command shown in [Example 11–2](#) adds an administered object to an LDAP object store representing a topic destination with lookup name `myTopic` and physical destination name `physTopic`. The command for adding a queue destination would be similar, except that the administered object type (`-t` option) would be `q` (for “queue destination”) instead of `t` (for “topic destination”).

EXAMPLE 11–2 Adding a Destination to an LDAP Object Store

```
imqobjmgr add
-l "cn=myTopic"
-j "java.naming.factory.initial=com.sun.jndi.ldap.LdapCtxFactory"
-j "java.naming.provider.url=ldap://mydomain.com:389/o=imq"
-j "java.naming.security.principal=uid=homerSimpson,ou=People,o=imq"
-j "java.naming.security.credentials=doh"
-j "java.naming.security.authentication=simple"
-t t
-o "imqDestinationName=physTopic"
```

[Example 11–3](#) shows the same command, but with the administered object stored in a Solaris file system instead of an LDAP server.

EXAMPLE 11–3 Adding a Destination to a File-System Object Store

```
imqobjmgr add
-l "cn=myTopic"
-j "java.naming.factory.initial=com.sun.jndi.fscontext.RefFSContextFactory"
-j "java.naming.provider.url=file:///home/foo/imq_admin_objects"
-t t
-o "imqDestinationName=physTopic"
```

Deleting Administered Objects

To delete an administered object from the object store, use the `imqobjmgr delete` subcommand, specifying the lookup name, type, and location of the object to be deleted. The command shown in [Example 11–4](#) deletes the object that was added in [“Adding a Destination” on page 201](#) above.

EXAMPLE 11–4 Deleting an Administered Object

```
imqobjmgr delete
-l "cn=myTopic"
-j "java.naming.factory.initial=com.sun.jndi.ldap.LdapCtxFactory"
-j "java.naming.provider.url=ldap://mydomain.com:389/o=imq"
-j "java.naming.security.principal=uid=homerSimpson,ou=People,o=imq"
-j "java.naming.security.credentials=doh"
-j "java.naming.security.authentication=simple"
-t t
```

Listing Administered Objects

You can use the `imqobjmgr list` subcommand to get a list of all administered objects in an object store or those of a specific type. [Example 11–5](#) shows how to list all administered objects on an LDAP server.

EXAMPLE 11–5 Listing All Administered Objects

```
imqobjmgr list
-j "java.naming.factory.initial=com.sun.jndi.ldap.LdapCtxFactory"
-j "java.naming.provider.url=ldap://mydomain.com:389/o=imq"
-j "java.naming.security.principal=uid=homerSimpson,ou=People,o=imq"
-j "java.naming.security.credentials=doh"
-j "java.naming.security.authentication=simple"
```

[Example 11–6](#) lists all queue destinations (type q).

EXAMPLE 11–6 Listing Administered Objects of a Specific Type

```
imqobjmgr list
-j "java.naming.factory.initial=com.sun.jndi.ldap.LdapCtxFactory"
-j "java.naming.provider.url=ldap://mydomain.com:389/o=imq"
-j "java.naming.security.principal=uid=homerSimpson,ou=People,o=imq"
-j "java.naming.security.credentials=doh"
-j "java.naming.security.authentication=simple"
-t q
```

Viewing Administered Object Information

The `imqobjmgr query` subcommand displays information about a specified administered object, identified by its lookup name and the attributes of the object store containing it.

[Example 11–7](#) displays information about an object whose lookup name is `cn=myTopic`.

EXAMPLE 11–7 Viewing Administered Object Information

```
imqobjmgr query
-l "cn=myTopic"
-j "java.naming.factory.initial=com.sun.jndi.ldap.LdapCtxFactory"
-j "java.naming.provider.url=ldap://mydomain.com:389/o=imq"
-j "java.naming.security.principal=uid=homerSimpson,ou=People,o=imq"
-j "java.naming.security.credentials=doh"
-j "java.naming.security.authentication=simple"
```

Modifying Administered Object Attributes

To modify the attributes of an administered object, use the `imqobjmgr update` subcommand. You supply the object's lookup name and location, and use the `-o` option to specify the new attribute values.

[Example 11–8](#) changes the value of the `imqReconnectAttempts` attribute for the queue connection factory that was added to the object store in [Example 11–1](#).

EXAMPLE 11–8 Modifying an Administered Object's Attributes

```
imqobjmgr update
-l "cn=myQCF"
-j "java.naming.factory.initial=com.sun.jndi.ldap.LdapCtxFactory"
-j "java.naming.provider.url=ldap://mydomain.com:389/o=imq"
-j "java.naming.security.principal=uid=homerSimpson,ou=People,o=imq"
-j "java.naming.security.credentials=doh"
-j "java.naming.security.authentication=simple"
-t qf
-o "imqReconnectAttempts=3"
```

Using Command Files

The `-i` option to the `imqobjmgr` command allows you to specify the name of a command file that uses Java property file syntax to represent all or part of the subcommand clause. This feature is especially useful for specifying object store attributes, which typically require a lot of typing and are likely to be the same across multiple invocations of `imqobjmgr`. Using a command file can also allow you to avoid exceeding the maximum number of characters allowed for the command line.

[Example 11–9](#) shows the general syntax for an Object Manager command file. Note that the version property is not a command line option: it refers to the version of the command file itself (not that of the Message Queue product) and must be set to the value `2.0`.

EXAMPLE 11–9 Object Manager Command File Syntax

```
version=2.0
cmdtype=[ add | delete | list | query | update ]
obj.lookupName=lookup name
objstore.attrs.objStoreAttrName1=value1
objstore.attrs.objStoreAttrName2=value2
. . .
objstore.attrs.objStoreAttrNameN=valueN
obj.type=[ q | t | cf | qf | tf | xcf | xqf | xtf | e ]
obj.attrs.objAttrName1=value1
obj.attrs.objAttrName2=value2
. . .
obj.attrs.objAttrNameN=valueN
```

As an example, consider the Object Manager command shown earlier in [Example 11–1](#), which adds a queue connection factory to an LDAP object store. This command can be encapsulated in a command file as shown in [Example 11–10](#). If the command file is named `MyCmdFile`, you can then execute the command with the command line

```
imqobjmgr -i MyCmdFile
```

EXAMPLE 11–10 Example Command File

```
version=2.0
cmdtype=add
obj.lookupName=cn=myQCF
objstore.attrs.java.naming.factory.initial=com.sun.jndi.ldap.LdapCtxFactory
objstore.attrs.java.naming.provider.url=ldap://mydomain.com:389/o=imq
objstore.attrs.java.naming.security.principal=uid=homerSimpson,ou=People,o=imq
objstore.attrs.java.naming.security.credentials=doh
objstore.attrs.java.naming.security.authentication=simple
obj.type=qf
obj.attrs.imqAddressList=mq://myHost:7272/jms
```

A command file can also be used to specify only part of the `imqobjmgr` subcommand clause, with the remainder supplied directly on the command line. For example, the command file shown in [Example 11–11](#) specifies only the attribute values for an LDAP object store.

EXAMPLE 11–11 Partial Command File

```
version=2.0
objstore.attrs.java.naming.factory.initial=com.sun.jndi.ldap.LdapCtxFactory
objstore.attrs.java.naming.provider.url=ldap://mydomain.com:389/o=imq
objstore.attrs.java.naming.security.principal=uid=homerSimpson,ou=People,o=imq
objstore.attrs.java.naming.security.credentials=doh
objstore.attrs.java.naming.security.authentication=simple
```

You could then use this command file to specify the object store in an `imqobjmgr` command while supplying the remaining options explicitly, as shown in [Example 11–12](#).

EXAMPLE 11–12 Using a Partial Command File

```
imqobjmgr add
-l "cn=myQCF"
-i MyCmdFile
-t qf
-o "imqAddressList=mq://myHost:7272/jms"
```

Additional examples of command files can be found at the following locations, depending on your platform:

Solaris `/usr/demo/imq/imqobjmgr`

Linux `/opt/sun/mq/examples/imqobjmgr`

Windows `IMQ_HOME\demo\imqobjmgr`

Monitoring Broker Operations

This chapter describes the tools you can use to monitor a broker and how you can get metrics data. The chapter has the following sections:

- “Monitoring Services” on page 209
- “Introduction to Monitoring Tools” on page 210
- “Configuring and Using Broker Logging” on page 212
- “Using the Command Utility to Display Metrics Interactively” on page 218
- “Using the JMX Administration API” on page 223
- “Using the Java ES Monitoring Console” on page 223
- “Using the Message-Based Monitoring API ” on page 224

Reference information on specific metrics is available in [Chapter 20, “Metrics Information Reference”](#)

Monitoring Services

The broker includes components for monitoring and diagnosing application and broker performance. These include the components and services shown in the following figure:

- Broker code that logs broker events.
- A metrics generator that provides.
 - The metrics generator provides information about broker activity, such as message flow in and out of the broker, the number of messages in broker memory and the memory they consume, the number of open connections, and the number of threads being used. The boolean broker property `imq.metrics.enabled` controls whether such information is logged and the `imq.metrics.interval` property specifies how often metrics information is generated.
- A logger component that writes out information to a number of output channels.
- A comprehensive set of Java Management Extensions (JMX) MBeans that expose broker resources using the JMX API

- Support for the Java ES Monitoring Framework
- A metrics message producer that sends JMS messages containing metrics information to topic destinations for consumption by JMS monitoring clients.

Broker properties for configuring the monitoring services are listed under “[Monitoring Properties](#)” on page 316.

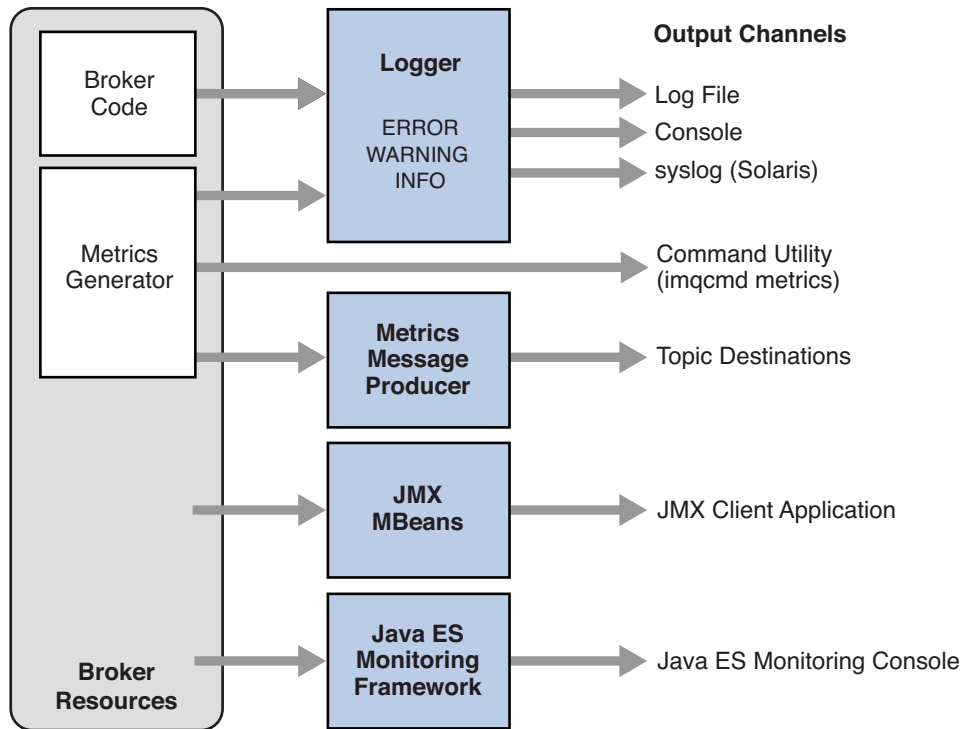


FIGURE 12-1 Monitoring Services Support

Introduction to Monitoring Tools

There are five tools (or interfaces) for monitoring Message Queue information, as described briefly below:

- **Log files** provide a long-term record of metrics data, but cannot easily be parsed.
- **The Command Utility** (`imqcmd metrics`) lets you interactively sample information tailored to your needs, but does not provide historical information or allow you to manipulate the data programmatically.

- **The Java Management Extensions (JMX) Administration API** lets you perform broker resource configuration and monitoring operations programmatically from within a Java application. You can write your own JMX administration application or use the standard Java Monitoring and Management Console (`jconsole`).
- **The Sun Java Enterprise System Monitoring Framework (JESMF) and Monitoring Console** offers a common, Web-based graphical interface shared with other Java ES components, but can monitor only a subset of all Message Queue entities and operations.
- **The Message-based Monitoring API** lets you extract metrics information from messages produced by the broker to metrics topic destinations. However, to use it, you must write a Message Queue client application to capture, analyze, and display the metrics data.

The following table compares the different tools.

TABLE 12-1 Benefits and Limitations of Metrics Monitoring Tools

Metrics Monitoring Tool	Benefits	Limitations
Log files	<ul style="list-style-type: none"> ■ Regular sampling ■ Creates a historical record 	<ul style="list-style-type: none"> ■ Local monitoring only ■ Data format difficult to read; no parsing tools ■ Need to configure broker properties; must shut down and restart broker to take effect ■ Broker metrics only; no destination or connection service metrics ■ No flexibility in selection of data ■ Same reporting interval for all metrics data; cannot be changed on the fly ■ Possible performance penalty if interval set too short
Command Utility (<code>imqcmd metrics</code>)	<ul style="list-style-type: none"> ■ Remote monitoring ■ Convenient for spot-checking ■ Data presented in easy-to-read tabular format ■ Easy to select specific data of interest ■ Reporting interval set in command option; can be changed on the fly 	<ul style="list-style-type: none"> ■ Difficult to analyze data programmatically ■ No single command gets all data ■ No historical record; difficult to see historical trends

TABLE 12-1 Benefits and Limitations of Metrics Monitoring Tools *(Continued)*

Metrics Monitoring Tool	Benefits	Limitations
JMX Administration API	<ul style="list-style-type: none"> ■ Remote monitoring ■ Data can be analyzed programmatically and presented in any format ■ Easy to select specific data of interest ■ Can use standard Java Monitoring and Management Console (<code>jconsole</code>) 	<ul style="list-style-type: none"> ■ Might need to configure broker's JMX support
Java ES Monitoring Console	<ul style="list-style-type: none"> ■ Web-based graphical interface ■ Data presented in easy-to-read format ■ Common interface shared with other JES components ■ No performance penalty; pulls data from broker's existing data monitoring infrastructure 	<ul style="list-style-type: none"> ■ Limited subset of data available ■ Data cannot be analyzed programmatically ■ No historical record; difficult to see historical trends
Message-based Monitoring API	<ul style="list-style-type: none"> ■ Remote monitoring ■ Data can be analyzed programmatically and presented in any format ■ Easy to select specific data of interest 	<ul style="list-style-type: none"> ■ Need to configure broker properties; must shut down and restart broker to take effect ■ Same reporting interval for all metrics data; cannot be changed on the fly

In addition to the differences shown in the table, each tool gathers a somewhat different subset of the metrics information generated by the broker. For information on which metrics data is gathered by each monitoring tool, see [Chapter 20, “Metrics Information Reference.”](#)

Configuring and Using Broker Logging

The Message Queue Logger takes information generated by broker code, a debugger, and a metrics generator and writes that information to a number of output channels: to standard output (the console), to a log file, and, on Solaris platforms, to the `syslog` daemon process. You can specify the type of information gathered by the Logger as well as the type of information the Logger writes to each of the output channels. For example, you can specify that you want metrics information written out to a log file.

This section describes the configuration and use of the Logger for monitoring broker activity. It includes the following topics:

- [“Logger Properties” on page 213](#)
- [“Log Message Format” on page 213](#)
- [“Default Logging Configuration” on page 214](#)
- [“Changing the Logging Configuration” on page 215](#)

Logger Properties

The `imq.log.file.dirpath` and `imq.log.file.filename` broker properties identify the log file to use and the `imq.log.console.stream` property specifies whether console output is directed to `stdout` or `stderr`.

The `imq.log.level` property controls the categories of metric information that the Logger gathers: `ERROR`, `WARNING`, or `INFO`. Each level includes those above it, so if you specify, for example, `WARNING` as the logging level, error messages will be logged as well.

There is also an `imq.destination.logDeadMsgs` property that specifies whether to log entries when dead messages are discarded or moved to the dead message queue.

The `imq.log.console.output` and `imq.log.file.output` properties control which of the specified categories the Logger writes to the console and the log file, respectively. In this case, however, the categories do *not* include those above them; so if you want, for instance, both errors and warnings written to the log file and informational messages to the console, you must explicitly set `imq.log.file.output` to `ERROR|WARNING` and `imq.log.console.output` to `INFO`.

On Solaris platforms another property, `imq.log.syslog.output`, specifies the categories of metric information to be written to the `syslog` daemon.

In the case of a log file, you can specify the point at which the file is closed and output is rolled over to a new file. Once the log file reaches a specified size (`imq.log.file.rolloverbytes`) or age (`imq.log.file.rolloversecs`), it is saved and a new log file created.

See [“Monitoring Properties” on page 316](#) for additional broker properties related to logging and subsequent sections for details about how to configure the Logger and how to use it to obtain performance information.

Log Message Format

A logged message consists of a time stamp, a message code, and the message itself. The volume of information included varies with the *logging level* you have set. The broker supports three logging levels: `ERROR`, `WARNING`, and `INFO` (see [Table 12–2](#)). Each level includes those above it (for example, `WARNING` includes `ERROR`).

TABLE 12-2 Logging Levels

Logging Level	Description
ERROR	Serious problems that could cause system failure
WARNING	Conditions that should be heeded but will not cause system failure
INFO	Metrics and other informational messages

The default logging level is INFO, so messages at all three levels are logged by default. The following is an example of an INFO message:

```
[13/Sep/2000:16:13:36 PDT] [B1004]: Starting the broker service
using tcp [25374,100] with min threads 50 and max threads of 500
```

You can change the time zone used in the time stamp by setting the broker configuration property `imq.log.timezone` (see [Table 16-10](#)).

Default Logging Configuration

A broker is automatically configured to save log output to a set of rolling log files. The log files are located in a directory identified by the instance name of the associated broker (see [Appendix A, “Platform-Specific Locations of Message Queue Data”](#)):

```
.../instances/instanceName/log
```

Note – For a broker whose life cycle is controlled by the Application Server, the log files are located in a subdirectory of the domain directory for the domain for which the broker was started:

```
.../appServerDomainDir/imq/instances/imqbroker/log
```

The log files are simple text files. The system maintains nine backup files named as follows, from earliest to latest:

```
log.txt
log_1.txt
log_2.txt
...
log_9.txt
```

By default, the log files are rolled over once a week. You can change this rollover interval, or the location or names of the log files, by setting appropriate configuration properties:

- To change the directory in which the log files are kept, set the property `imq.log.file.dirpath` to the desired path.

- To change the root name of the log files from `log` to something else, set the `imq.log.file.filename` property.
- To change the frequency with which the log files are rolled over, set the property `imq.log.file.rolloversecs`.

See [Table 16–10](#) for further information on these properties.

Changing the Logging Configuration

Log-related properties are described in [Table 16–10](#).

▼ To Change the Logger Configuration for a Broker

- 1 Set the logging level.
- 2 Set the output channel (file, console, or both) for one or more logging categories.
- 3 If you log output to a file, configure the rollover criteria for the file.

You complete these steps by setting Logger properties. You can do this in one of two ways:

- Change or add Logger properties in the `config.properties` file for a broker before you start the broker.
- Specify Logger command line options in the `imqbrokerd` command that starts the broker. You can also use the broker option `-D` to change Logger properties (or *any* broker property).

Options passed on the command line override properties specified in the broker instance configuration files. The following `imqbrokerd` options affect logging:

<code>-metrics interval</code>	Logging interval for broker metrics, in seconds
<code>-loglevel level</code>	Logging level (ERROR, WARNING, INFO, or NONE)
<code>-silent</code>	Silent mode (no logging to console)
<code>-tty</code>	Log all messages to console

The following sections describe how you can change the default configuration in order to do the following:

- Change the output channel (the destination of log messages)
- Change rollover criteria

Changing the Output Channel

By default, error and warning messages are displayed on the terminal as well as being logged to a log file. (On Solaris, error messages are also written to the system's `syslog` daemon.)

You can change the output channel for log messages in the following ways:

- To have *all* log categories (for a given level) output displayed on the screen, use the `-tty` option to the `imqbrokerd` command.
- To prevent log output from being displayed on the screen, use the `-silent` option to the `imqbrokerd` command.
- Use the `imq.log.file.output` property to specify which categories of logging information should be written to the log file. For example,

```
imq.log.file.output=ERROR
```

- Use the `imq.log.console.output` property to specify which categories of logging information should be written to the console. For example,

```
imq.log.console.output=INFO
```

- On Solaris, use the `imq.log.syslog.output` property to specify which categories of logging information should be written to Solaris `syslog`. For example,

```
imq.log.syslog.output=NONE
```

Note – Before changing Logger output channels, you must make sure that logging is set at a level that supports the information you are mapping to the output channel. For example, if you set the logging level to `ERROR` and then set the `imq.log.console.output` property to `WARNING`, no messages will be logged because you have not enabled the logging of `WARNING` messages.

Changing Log File Rollover Criteria

There are two criteria for rolling over log files: time and size. The default is to use a time criteria and roll over files every seven days.

- To change the time interval, you need to change the property `imq.log.file.rolloversecs`. For example, the following property definition changes the time interval to ten days:

```
imq.log.file.rolloversecs=864000
```

- To change the rollover criteria to depend on file size, you need to set the `imq.log.file.rolloverbytes` property. For example, the following definition directs the broker to rollover files after they reach a limit of 500,000 bytes

```
imq.log.file.rolloverbytes=500000
```

If you set both the time-related and the size-related rollover properties, the first limit reached will trigger the rollover. As noted before, the broker maintains up to nine rollover files.

You can set or change the log file rollover properties when a broker is running. To set these properties, use the `imqcmd update bkr` command.

Sending Metrics Data to Log Files

This section describes the procedure for using broker log files to report metrics information. For general information on configuring the Logger, see [“Configuring and Using Broker Logging” on page 212](#).

▼ To Use Log Files to Report Metrics Information

1 Configure the broker’s metrics generation capability:

a. Confirm `imq.metrics.enabled=true`

Generation of metrics for logging is turned on by default.

b. Set the metrics generation interval to a convenient number of seconds.

```
imq.metrics.interval=interval
```

This value can be set in the `config.properties` file or using the `-metrics interval` command line option when starting up the broker.

2 Confirm that the Logger gathers metrics information:

```
imq.log.level=INFO
```

This is the default value. This value can be set in the `config.properties` file or using the `-loglevel level` command line option when starting up the broker.

3 Confirm that the Logger is set to write metrics information to the log file:

```
imq.log.file.output=INFO
```

This is the default value. It can be set in the `config.properties` file.

4 Start up the broker.

The following shows sample broker metrics output to the log file:

```
[21/Jul/2004:11:21:18 PDT]
Connections: 0      JVM Heap: 8323072 bytes (7226576 free) Threads: 0 (14-1010)
  In: 0 msgs (0bytes) 0 pkts (0 bytes)
  Out: 0 msgs (0bytes) 0 pkts (0 bytes)
  Rate In: 0 msgs/sec (0 bytes/sec) 0 pkts/sec (0 bytes/sec)
  Rate Out: 0 msgs/sec (0 bytes/sec) 0 pkts/sec (0 bytes/sec)
```

For reference information about metrics data, see [Chapter 20, “Metrics Information Reference”](#)

Logging Dead Messages

You can monitor physical destinations by enabling dead message logging for a broker. You can log dead messages whether or not you are using a dead message queue.

If you enable dead message logging, the broker logs the following types of events:

- A physical destination exceeded its maximum size.
- The broker removed a message from a physical destination, for a reason such as the following:
 - The destination size limit has been reached.
 - The message time to live expired.
 - The message is too large.
 - An error occurred when the broker attempted to process the message.

If a dead message queue is in use, logging also includes the following types of events:

- The broker moved a message to the dead message queue.
- The broker removed a message from the dead message queue and discarded it.

The following is an example of the log format for dead messages:

```
[29/Mar/2006:15:35:39 PST] [B1147]: Message 8-129.145.180.87(e7:6b:dd:5d:98:aa)-
35251-1143675279400 from destination Q:q0 has been placed on the DMQ because
[B0053]: Message on destination Q:q0 Expired: expiration time 1143675279402,
arrival time 1143675279401, JMSTimestamp 1143675279400
```

Dead message logging is disabled by default. To enable it, set the broker attribute `imq.destination.logDeadMsgs`.

Using the Command Utility to Display Metrics Interactively

A Message Queue broker can report metrics of the following types:

- **Java Virtual Machine (JVM) metrics.** Information about the JVM heap size.
- **Brokerwide metrics.** Information about messages stored in a broker, message flows into and out of a broker, and memory use. Messages are tracked in terms of numbers of messages and numbers of bytes.
- **Connection Service metrics.** Information about connections and connection thread resources, and information about message flows for a particular connection service.
- **Destination metrics.** Information about message flows into and out of a particular physical destination, information about a physical destination's consumers, and information about memory and disk space usage.

The `imqcmd` command can obtain metrics information for the broker as a whole, for individual connection services, and for individual physical destinations. To obtain metrics data, you generally use the `metrics` subcommand of `imqcmd`. Metrics data is written at an interval you specify, or the number of times you specify, to the console screen.

You can also use the `query` subcommand to view similar data that also includes configuration information. See [“`imqcmd query`” on page 222](#) for more information.

imqcmd metrics

The syntax and options of `imqcmd metrics` are shown in [Table 12–3](#) and [Table 12–4](#), respectively.

TABLE 12–3 `imqcmd metrics` Subcommand Syntax

Subcommand Syntax	Metrics Data Provided
<pre>metrics bkr [-b <i>hostName:portNumber</i>] [-m <i>metricType</i>] [-int <i>interval</i>] [-msp <i>numSamples</i>]</pre>	Displays broker metrics for the default broker or a broker at the specified host and port.
<pre>metrics svc -n <i>serviceName</i> [-b <i>hostName:portNumber</i>] [-m <i>metricType</i>] [-int <i>interval</i>] [-msp <i>numSamples</i>]</pre>	Displays metrics for the specified service on the default broker or on a broker at the specified host and port.
<pre>metrics dst -t <i>destType</i> -n <i>destName</i> [-b <i>hostName:portNumber</i>] [-m <i>metricType</i>] [-int <i>interval</i>] [-msp <i>numSamples</i>]</pre>	Displays metrics information for the physical destination of the specified type and name.

TABLE 12–4 `imqcmd metrics` Subcommand Options

Subcommand Options	Description
<code>-b <i>hostName:portNumber</i></code>	Specifies the hostname and port of the broker for which metrics data is reported. The default is <code>localhost:7676</code> .
<code>-int <i>interval</i></code>	Specifies the interval (in seconds) at which to display the metrics. The default is 5 seconds.

TABLE 12-4 imqcmd metrics Subcommand Options (Continued)

Subcommand Options	Description
-m <i>metricType</i>	Specifies the type of metric to display: ttl Displays metrics on messages and packets flowing into and out of the broker, service, or destination (default metric type). rts Displays metrics on rate of flow of messages and packets into and out of the broker, connection service, or destination (per second). cxn Displays connections, virtual memory heap, and threads (brokers and connection services only). con Displays consumer-related metrics (destinations only). dsk Displays disk usage metrics (destinations only).
-msp <i>numSamples</i>	Specifies the number of samples displayed in the output. The default is an unlimited number (infinite).
-n <i>destName</i>	Specifies the name of the physical destination (if any) for which metrics data is reported. There is no default.
-n <i>serviceName</i>	Specifies the connection service (if any) for which metrics data is reported. There is no default.
-t <i>destType</i>	Specifies the type (queue or topic) of the physical destination (if any) for which metrics data is reported. There is no default.

▼ To Use the metrics Subcommand

- 1 Start the broker for which metrics information is desired.
See “Starting Brokers” on page 68.
- 2 Issue the appropriate `imqcmd metrics` subcommand and options as shown in [Table 12-3](#) and [Table 12-4](#).

Metrics Outputs: imqcmd metrics

This section contains examples of output for the `imqcmd metrics` subcommand. The examples show brokerwide, connection service, and physical destination metrics.

Brokerwide Metrics

To get the rate of message and packet flow into and out of the broker at 10 second intervals, use the `metrics bkr` subcommand:

```
imqcmd metrics bkr -m rts -int 10 -u admin
```

This command produces output similar to the following (see data descriptions in [Table 20–2](#)):

Msgs/sec		Msg Bytes/sec		Pkts/sec		Pkt Bytes/sec	
In	Out	In	Out	In	Out	In	Out
0	0	27	56	0	0	38	66
10	0	7365	56	10	10	7457	1132
0	0	27	56	0	0	38	73
0	10	27	7402	10	20	1400	8459
0	0	27	56	0	0	38	73

Connection Service Metrics

To get cumulative totals for messages and packets handled by the `jms` connection service, use the `metrics svc` subcommand:

```
imqcmd metrics svc -n jms -m ttl -u admin
```

This command produces output similar to the following (see data descriptions in [Table 20–3](#)):

Msgs		Msg Bytes		Pkts		Pkt Bytes	
In	Out	In	Out	In	Out	In	Out
164	100	120704	73600	282	383	135967	102127
657	100	483552	73600	775	876	498815	149948

Physical Destination Metrics

To get metrics information about a physical destination, use the `metrics dst` subcommand:

```
imqcmd metrics dst -t q -n XQueue -m ttl -u admin
```

This command produces output similar to the following (see data descriptions in [Table 20–4](#)):

Msgs		Msg Bytes		Msg Count			Total Msg Bytes (k)			Largest
In	Out	In	Out	Current	Peak	Avg	Current	Peak	Avg	Msg (k)
200	200	147200	147200	0	200	0	0	143	71	0
300	200	220800	147200	100	200	10	71	143	64	0
300	300	220800	220800	0	200	0	0	143	59	0

To get information about a physical destination's consumers, use the following `metrics dst` subcommand:

```
imqcmd metrics dst -t q -n SimpleQueue -m con -u admin
```

This command produces output similar to the following (see data descriptions in [Table 20–4](#)):

Active Consumers			Backup Consumers			Msg Count		
Current	Peak	Avg	Current	Peak	Avg	Current	Peak	Avg
1	1	0	0	0	0	944	1000	525

imqcmd query

The syntax and options of `imqcmd query` are shown in [Table 12–5](#) along with a description of the metrics data provided by the command.

TABLE 12–5 `imqcmd query` Subcommand Syntax

Subcommand Syntax	Metrics Data Provided
<code>query bkr</code> [-b <i>hostName: portNumber</i>]	Information on the current number of messages and message bytes stored in broker memory and persistent store (see “Viewing Broker Information” on page 91).
or	
<code>query svc -n <i>serviceName</i></code> [-b <i>hostName:portNumber</i>]	Information on the current number of allocated threads and number of connections for a specified connection service (see “Viewing Connection Service Information” on page 99).
or	
<code>query dst -t <i>destType</i></code> -n <i>destName</i> [-b <i>hostName:portNumber</i>]	Information on the current number of producers, active and backup consumers, and messages and message bytes stored in memory and persistent store for a specified destination (see “Viewing Physical Destination Information” on page 112).

Note – Because of the limited metrics data provided by `imqcmd query`, this tool is not represented in the tables presented in [Chapter 20, “Metrics Information Reference.”](#)

Using the JMX Administration API

The broker implements a comprehensive set of Java Management Extensions (JMX) MBeans that represent the broker's manageable resources. Using the JMX API, you can access these MBeans to perform broker configuration and monitoring operations programmatically from within a Java application.

In this way, the MBeans provide a Java application access to data values representing static or dynamic properties of a broker, connection, destination, or other resource. The application can also receive notifications of state changes or other significant events affecting the resource.

JMX-based administration provides dynamic, fine grained, programmatic access to the broker. You can use this kind of administration in a number of ways.

- You can include JMX code in your JMS client application to monitor application performance and, based on the results, to reconfigure the Message Queue resources you use to improve performance.
- You can write JMX client applications that monitor the broker to identify use patterns and performance problems, and you can use the JMX API to reconfigure the broker to optimize performance.
- You can write a JMX client application to automate regular maintenance tasks.
- You can write a JMX client application that constitutes your own version of the Command utility (`imqcmd`), and you can use it instead of `imqcmd`.
- You can use the standard Java Monitoring and Management Console (`jconsole`) that can provide access to the broker's MBeans.

For information on JMX infrastructure and configuring the broker's JMX support, see [Appendix D, “JMX Support.”](#) To manage a Message Queue broker using the JMX architecture, see *Sun Java System Message Queue 4.2 Developer's Guide for JMX Clients*.

Using the Java ES Monitoring Console

Message Queue supports the Sun Java System Monitoring Framework (JESMF), which allows Java Enterprise System (Java ES) components to be monitored using a common graphical interface, the Sun Java System Monitoring Console. Administrators can use the Monitoring Console to view performance statistics, create rules for automatic monitoring, and acknowledge alarms. If you are running Message Queue along with other Java ES components, you may find it more convenient to use a single interface to manage all of them.

The Java ES Monitoring Framework defines a common data model, the Common Monitoring Model (CMM), to be used by all Java ES component products. This model enables a centralized and uniform view of all Java ES components. Message Queue exposes the following objects through the Common Monitoring Model:

- The installed product
- The broker instance name
- The broker Port Mapper
- Each connection service
- Each physical destination
- The persistent data store
- The user repository

Each of these objects is mapped to a CMM object whose attributes can be monitored using the Java ES Monitoring Console. The reference tables in [Chapter 21, “JES Monitoring Framework Reference,”](#) identify those attributes that are available for JESMF monitoring. For detailed information about the mapping of Message Queue objects to CMM objects, see the *Sun Java Enterprise System Monitoring Guide*.

To enable JESMF monitoring, you must do the following:

1. Enable and configure the Monitoring Framework for all of your monitored components, as described in the *Sun Java Enterprise System Monitoring Guide*.
2. Install the Monitoring Console on a separate host, start the master agent, and then start the Web server, as described in the *Sun Java Enterprise System Monitoring Guide*.

Using the Java ES Monitoring Framework will not affect broker performance, because all the work of gathering metrics is done by the Monitoring Framework, which pulls data from the broker's existing data monitoring infrastructure.

For information on metric information provided by the Java ES Monitoring Framework, see [Chapter 21, “JES Monitoring Framework Reference.”](#)

Using the Message-Based Monitoring API

Message Queue provides a Metrics Message Producer, which receives information from the Metrics Generator at regular intervals and writes the information into *metrics messages*. The Metrics Message Producer then sends these messages to one of a number of metric topic destinations, depending on the type of metric information contained in the messages.

You can access this metrics information by writing a client application that subscribes to the metrics topic destinations, consumes the messages in these destinations, and processes the metrics information contained in the messages. This allows you to create custom monitoring tools to support messaging applications. For details of the metric quantities reported in each type of metrics message, see [Chapter 4, “Using the Metrics Monitoring API,”](#) in *Sun Java System Message Queue 4.2 Developer's Guide for Java Clients*

There are five metrics topic destinations, whose names are shown in [Table 12–6](#), along with the type of metrics messages delivered to each destination.

TABLE 12-6 Metrics Topic Destinations

Topic Name	Description
<code>mq.metrics.broker</code>	Broker metrics
<code>mq.metrics.jvm</code>	Java Virtual Machine metrics
<code>mq.metrics.destination_list</code>	List of destinations and their types
<code>mq.metrics.destination.queue.queueName</code>	Destination metrics for queue <i>queueName</i>
<code>mq.metrics.destination.topic.topicName</code>	Destination metrics for topic <i>topicName</i>

The broker properties `mq.metrics.topic.enabled` and `mq.metrics.topic.interval` control, respectively, whether messages are sent to metric topic destinations and how often. The `mq.metrics.topic.timetolive` and `mq.metrics.topic.persist` properties specify the lifetime of such messages and whether they are persistent.

Besides the information contained in the body of a metrics message, the header of each message includes properties that provide the following additional information:

- The message type
- The address (host name and port number) of the broker that sent the message
- The time the metric sample was taken

These properties are useful to client applications that process metrics messages of different types or from different brokers.

Setting Up Message-Based Monitoring

This section describes the procedure for using the message-based monitoring capability to gather metrics information. The procedure includes both client development and administration tasks.

▼ To Set Up Message-based Monitoring

1 Write a metrics monitoring client.

See the *Message Queue Developer's Guide for Java Clients* for instructions on programming clients that subscribe to metrics topic destinations, consume metrics messages, and extract the metrics data from these messages.

2 Configure the broker's Metrics Message Producer by setting broker property values in the `config.properties` file:

a. Enable metrics message production.

Set `mq.metrics.topic.enabled=true`

The default value is `true`.

b. Set the interval (in seconds) at which metrics messages are generated.

Set `imq.metrics.topic.interval=interval`.

The default is 60 seconds.

c. Specify whether you want metrics messages to be persistent (that is, whether they will survive a broker failure).

Set `imq.metrics.topic.persist`.

The default is `false`.

d. Specify how long you want metrics messages to remain in their respective destinations before being deleted.

Set `imq.metrics.topic.timetolive`.

The default value is 300 seconds.

3 Set any access control you desire on metrics topic destinations.

See the discussion in [“Security and Access Considerations” on page 226](#) below.

4 Start up your metrics monitoring client.

When consumers subscribe to a metrics topic, the metrics topic destination will automatically be created. Once a metrics topic has been created, the broker's metrics message producer will begin sending metrics messages to the metrics topic.

Security and Access Considerations

There are two reasons to restrict access to metrics topic destinations:

- Metrics data might include sensitive information about a broker and its resources.
- Excessive numbers of subscriptions to metrics topic destinations might increase broker overhead and negatively affect performance.

Because of these considerations, it is advisable to restrict access to metrics topic destinations.

Monitoring clients are subject to the same authentication and authorization control as any other client. Only users maintained in the Message Queue user repository are allowed to connect to the broker.

You can provide additional protections by restricting access to specific metrics topic destinations through an access control file, as described in [“User Authorization” on page 153](#).

For example, the following entries in an `accesscontrol.properties` file will deny access to the `imq.metrics.broker` metrics topic to everyone except `user1` and `user2`.

```
topic.mq.metrics.broker.consume.deny.user=*  
topic.mq.metrics.broker.consume.allow.user=user1,user2
```

The following entries will only allow users user3 to monitor topic t1.

```
topic.mq.metrics.destination.topic.t1.consume.deny.user=*  
topic.mq.metrics.destination.topic.t1.consume.allow.user=user3
```

Depending on the sensitivity of metrics data, you can also connect your metrics monitoring client to a broker using an encrypted connection. For information on using encrypted connections, see [“Message Encryption” on page 159](#).

Metrics Outputs: Metrics Messages

The metrics data outputs you get using the message-based monitoring API is a function of the metrics monitoring client you write. You are limited only by the data provided by the metrics generator in the broker. For a complete list of this data, see [Chapter 20, “Metrics Information Reference.”](#)

Analyzing and Tuning a Message Service

This chapter covers a number of topics about how to analyze and tune a Message Queue service to optimize the performance of your messaging applications. It includes the following topics:

- [“About Performance” on page 229](#)
- [“Factors Affecting Performance” on page 232](#)
- [“Adjusting Configuration To Improve Performance” on page 242](#)

About Performance

This section provides some background information on performance tuning.

The Performance Tuning Process

The performance you get out of a messaging application depends on the interaction between the application and the Message Queue service. Hence, maximizing performance requires the combined efforts of both the application developer and the administrator.

The process of optimizing performance begins with application design and continues on through tuning the message service after the application has been deployed. The performance tuning process includes the following stages:

- Defining performance requirements for the application
- Designing the application taking into account factors that affect performance (especially tradeoffs between reliability and performance)
- Establishing baseline performance measures
- Tuning or reconfiguring the message service to optimize performance

The process outlined above is often iterative. During deployment of the application, a Message Queue administrator evaluates the suitability of the message service for the application’s general performance requirements. If the benchmark testing meets these requirements, the

administrator can tune the system as described in this chapter. However, if benchmark testing does not meet performance requirements, a redesign of the application might be necessary or the deployment architecture might need to be modified.

Aspects of Performance

In general, performance is a measure of the speed and efficiency with which a message service delivers messages from producer to consumer. However, there are several different aspects of performance that might be important to you, depending on your needs.

Connection Load	The number of message producers, or message consumers, or the number of concurrent connections a system can support.
Message throughput	The number of messages or message bytes that can be pumped through a messaging system per second.
Latency	The time it takes a particular message to be delivered from message producer to message consumer.
Stability	The overall availability of the message service or how gracefully it degrades in cases of heavy load or failure.
Efficiency	The efficiency of message delivery; a measure of message throughput in relation to the computing resources employed.

These different aspects of performance are generally interrelated. If message throughput is high, that means messages are less likely to be backlogged in the broker, and as a result, latency should be low (a single message can be delivered very quickly). However, latency can depend on many factors: the speed of communication links, broker processing speed, and client processing speed, to name a few.

In any case, the aspects of performance that are most important to you generally depends on the requirements of a particular application.

Benchmarks

Benchmarking is the process of creating a test suite for your messaging application and of measuring message throughput or other aspects of performance for this test suite.

For example, you could create a test suite by which some number of producing clients, using some number of connections, sessions, and message producers, send persistent or nonpersistent messages of a standard size to some number of queues or topics (all depending on your messaging application design) at some specified rate. Similarly, the test suite includes some number of consuming clients, using some number of connections, sessions, and message

consumers (of a particular type) that consume the messages in the test suite's physical destinations using a particular acknowledgment mode.

Using your standard test suite you can measure the time it takes between production and consumption of messages or the average message throughput rate, and you can monitor the system to observe connection thread usage, message storage data, message flow data, and other relevant metrics. You can then ramp up the rate of message production, or the number of message producers, or other variables, until performance is negatively affected. The maximum throughput you can achieve is a benchmark for your message service configuration.

Using this benchmark, you can modify some of the characteristics of your test suite. By carefully controlling all the factors that might have an effect on performance (see [“Application Design Factors Affecting Performance” on page 234](#)), you can note how changing some of these factors affects the benchmark. For example, you can increase the number of connections or the size of messages five-fold or ten-fold, and note the effect on performance.

Conversely, you can keep application-based factors constant and change your broker configuration in some controlled way (for example, change connection properties, thread pool properties, JVM memory limits, limit behaviors, file-based versus JDBC-based persistence, and so forth) and note how these changes affect performance.

This benchmarking of your application provides information that can be valuable when you want to increase the performance of a deployed application by tuning your message service. A benchmark allows the effect of a change or a set of changes to be more accurately predicted.

As a general rule, benchmarks should be run in a controlled test environment and for a long enough period of time for your message service to stabilize. (Performance is negatively affected at startup by the just-in-time compilation that turns Java code into machine code.)

Baseline Use Patterns

Once a messaging application is deployed and running, it is important to establish baseline use patterns. You want to know when peak demand occurs and you want to be able to quantify that demand. For example, demand normally fluctuates by number of end users, activity levels, time of day, or all of these.

To establish baseline use patterns you need to monitor your message service over an extended period of time, looking at data such as the following:

- Number of connections
- Number of messages stored in the broker (or in particular physical destinations)
- Message flows into and out of a broker (or particular physical destinations)
- Numbers of active consumers

You can also use average and peak values provided in metrics data.

It is important to check these baseline metrics against design expectations. By doing so, you are checking that client code is behaving properly: for example, that connections are not being left open or that consumed messages are not being left unacknowledged. These coding errors consume broker resources and could significantly affect performance.

The base-line use patterns help you determine how to tune your system for optimal performance. For example:

- If one physical destination is used significantly more than others, you might want to set higher message memory limits on that physical destination than on others, or to adjust limit behaviors accordingly.
- If the number of connections needed is significantly greater than allowed by the maximum thread pool size, you might want to increase the thread pool size or adopt a shared thread model.
- If peak message flows are substantially greater than average flows, that might influence the limit behaviors you employ when memory runs low.

In general, the more you know about use patterns, the better you are able to tune your system to those patterns and to plan for future needs.

Factors Affecting Performance

Message latency and message throughput, two of the main performance indicators, generally depend on the time it takes a typical message to complete various steps in the message delivery process. These steps are shown below for the case of a persistent, reliably delivered message. The steps are described following the illustration.

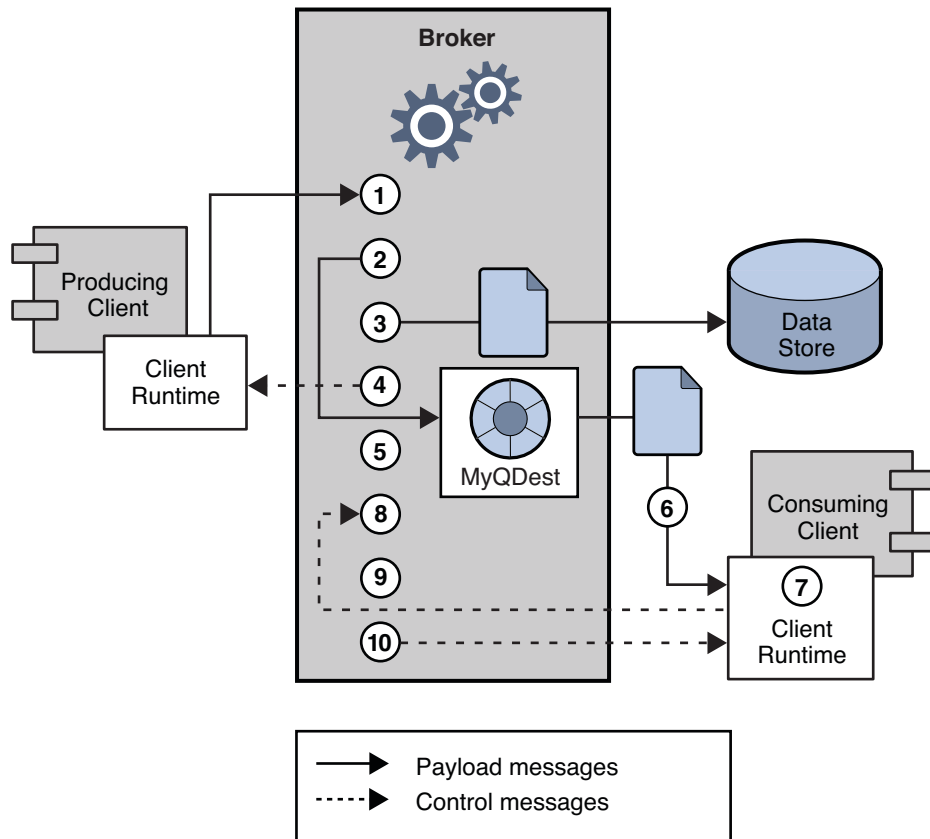


FIGURE 13-1 Message Delivery Through a Message Queue Service

Message Delivery Steps

1. The message is delivered from producing client to broker.
2. The broker reads in the message.
3. The message is placed in persistent storage (for reliability).
4. The broker confirms receipt of the message (for reliability).
5. The broker determines the routing for the message.
6. The broker writes out the message.
7. The message is delivered from broker to consuming client.
8. The consuming client acknowledges receipt of the message (for reliability).
9. The broker processes client acknowledgment (for reliability).
10. The broker confirms that client acknowledgment has been processed.

Since these steps are sequential, any one of them can be a potential bottleneck in the delivery of messages from producing clients to consuming clients. Most of the steps depend on physical

characteristics of the messaging system: network bandwidth, computer processing speeds, message service architecture, and so forth. Some, however, also depend on characteristics of the messaging application and the level of reliability it requires.

The following subsections discuss the effect of both application design factors and messaging system factors on performance. While application design and messaging system factors closely interact in the delivery of messages, each category is considered separately.

Application Design Factors Affecting Performance

Application design decisions can have a significant effect on overall messaging performance.

The most important factors affecting performance are those that affect the reliability of message delivery. Among these are the following:

- [“Delivery Mode \(Persistent/Nonpersistent Messages\)” on page 235](#)
- [“Use of Transactions” on page 235](#)
- [“Acknowledgment Mode” on page 236](#)
- [“Durable and Nondurable Subscriptions” on page 237](#)

Other application design factors affecting performance are the following:

- [“Use of Selectors \(Message Filtering\)” on page 237](#)
- [“Message Size” on page 237](#)
- [“Message Body Type” on page 238](#)

The sections that follow describe the effect of each of these factors on messaging performance. As a general rule, there is a tradeoff between performance and reliability: factors that increase reliability tend to decrease performance.

[Table 13–1](#) shows how the various application design factors generally affect messaging performance. The table shows two scenarios—one high-reliability, low-performance, and one high-performance, low-reliability—and the choices of application design factors that characterize each. Between these extremes, there are many choices and tradeoffs that affect both reliability and performance.

TABLE 13–1 Comparison of High-Reliability and High-Performance Scenarios

Application Design Factor	High-Reliability, Low-Performance Scenario	High-Performance, Low-Reliability Scenario
Delivery mode	Persistent messages	Nonpersistent messages
Use of transactions	Transacted sessions	No transactions
Acknowledgment mode	AUTO_ACKNOWLEDGE or CLIENT_ACKNOWLEDGE	DUPS_OK_ACKNOWLEDGE

TABLE 13–1 Comparison of High-Reliability and High-Performance Scenarios (Continued)

Application Design Factor	High-Reliability, Low-Performance Scenario	High-Performance, Low-Reliability Scenario
Durable/nondurable subscriptions	Durable subscriptions	Nondurable subscriptions
Use of selectors	Message filtering	No message filtering
Message size	Large number of small messages	Small number of large messages
Message body type	Complex body types	Simple body types

Delivery Mode (Persistent/Nonpersistent Messages)

Persistent messages guarantee message delivery in case of broker failure. The broker stores the message in a persistent store until all intended consumers acknowledge they have consumed the message.

Broker processing of persistent messages is slower than for nonpersistent messages for the following reasons:

- A broker must reliably store a persistent message so that it will not be lost should the broker fail.
- The broker must confirm receipt of each persistent message it receives. Delivery to the broker is guaranteed once the method producing the message returns without an exception.
- Depending on the client acknowledgment mode, the broker might need to confirm a consuming client's acknowledgment of a persistent message.

For both queues and topics with durable subscribers, performance was approximately 40% faster for nonpersistent messages. We obtained these results using 10k-sized messages and `AUTO_ACKNOWLEDGE` mode.

Use of Transactions

A transaction is a guarantee that all messages produced in a transacted session and all messages consumed in a transacted session will be either processed or not processed (rolled back) as a unit.

Message Queue supports both local and distributed transactions.

A message produced or acknowledged in a transacted session is slower than in a nontransacted session for the following reasons:

- Additional information must be stored with each produced message.
- In some situations, messages in a transaction are stored when normally they would not be (for example, a persistent message delivered to a topic destination with no subscriptions would normally be deleted, however, at the time the transaction is begun, information about subscriptions is not available).

- Information on the consumption and acknowledgment of messages within a transaction must be stored and processed when the transaction is committed.

Note – To improve performance, Message Queue message brokers are configured by default to use a memory-mapped file to store transaction data. On file systems that do not support memory-mapped files, you can disable this behavior by setting the broker property `imq.persist.file.transaction.memorymappedfile.enabled` to `false`.

Acknowledgment Mode

One mechanism for ensuring the reliability of JMS message delivery is for a client to acknowledge consumption of messages delivered to it by the Message Queue broker.

If a session is closed without the client acknowledging the message or if the broker fails before the acknowledgment is processed, the broker redelivers that message, setting a `JMSRedelivered` flag.

For a nontransacted session, the client can choose one of three acknowledgment modes, each of which has its own performance characteristics:

- `AUTO_ACKNOWLEDGE`. The system automatically acknowledges a message once the consumer has processed it. This mode guarantees at most one redelivered message after a provider failure.
- `CLIENT_ACKNOWLEDGE`. The application controls the point at which messages are acknowledged. All messages processed in that session since the previous acknowledgment are acknowledged. If the broker fails while processing a set of acknowledgments, one or more messages in that group might be redelivered.
- `DUPS_OK_ACKNOWLEDGE`. This mode instructs the system to acknowledge messages in a lazy manner. Multiple messages can be redelivered after a provider failure.

(Using `CLIENT_ACKNOWLEDGE` mode is similar to using transactions, except there is no guarantee that all acknowledgments will be processed together if a provider fails during processing.)

Acknowledgment mode affects performance for the following reasons:

- Extra control messages between broker and client are required in `AUTO_ACKNOWLEDGE` and `CLIENT_ACKNOWLEDGE` modes. The additional control messages add additional processing overhead and can interfere with JMS payload messages, causing processing delays.
- In `AUTO_ACKNOWLEDGE` and `CLIENT_ACKNOWLEDGE` modes, the client must wait until the broker confirms that it has processed the client's acknowledgment before the client can consume additional messages. (This broker confirmation guarantees that the broker will not inadvertently redeliver these messages.)
- The Message Queue persistent store must be updated with the acknowledgment information for all persistent messages received by consumers, thereby decreasing performance.

Durable and Nondurable Subscriptions

Subscribers to a topic destination fall into two categories, those with durable and nondurable subscriptions.

Durable subscriptions provide increased reliability but slower throughput, for the following reasons:

- The Message Queue message service must persistently store the list of messages assigned to each durable subscription so that should a broker fail, the list is available after recovery.
- Persistent messages for durable subscriptions are stored persistently, so that should a broker fail, the messages can still be delivered after recovery, when the corresponding consumer becomes active. By contrast, persistent messages for nondurable subscriptions are not stored persistently (should a broker fail, the corresponding consumer connection is lost and the message would never be delivered).

We compared performance for durable and nondurable subscribers in two cases: persistent and nonpersistent 10k-sized messages. Both cases use `AUTO_ACKNOWLEDGE` acknowledgment mode. We found an effect on performance only in the case of persistent messages which slowed durables by about 30%

Use of Selectors (Message Filtering)

Application developers often want to target sets of messages to particular consumers. They can do so either by targeting each set of messages to a unique physical destination or by using a single physical destination and registering one or more selectors for each consumer.

A selector is a string requesting that only messages with property values that match the string are delivered to a particular consumer. For example, the selector `NumberOfOrders >1` delivers only the messages with a `NumberOfOrders` property value of 2 or more.

Creating consumers with selectors lowers performance (as compared to using multiple physical destinations) because additional processing is required to handle each message. When a selector is used, it must be parsed so that it can be matched against future messages. Additionally, the message properties of each message must be retrieved and compared against the selector as each message is routed. However, using selectors provides more flexibility in a messaging application.

Message Size

Message size affects performance because more data must be passed from producing client to broker and from broker to consuming client, and because for persistent messages a larger message must be stored.

However, by batching smaller messages into a single message, the routing and processing of individual messages can be minimized, providing an overall performance gain. In this case, information about the state of individual messages is lost.

In our tests, which compared throughput in kilobytes per second for 1k, 10k, and 100k-sized messages to a queue destination and `AUTO_ACKNOWLEDGE` acknowledgment mode, we found that nonpersistent messaging was about 50% faster for 1k messages, about 20% faster for 10k messages, and about 5% faster for 100k messages. The size of the message affected performance significantly for both persistent and nonpersistent messages. 100k messages are about 10 times faster than 10k, and 10k are about 5 times faster than 1k.

Message Body Type

JMS supports five message body types, shown below roughly in the order of complexity:

- `BytesMessage` contains a set of bytes in a format determined by the application.
- `TextMessage` is a simple Java string.
- `StreamMessage` contains a stream of Java primitive values.
- `MapMessage` contains a set of name-value pairs.
- `ObjectMessage` contains a Java serialized object.

While, in general, the message type is dictated by the needs of an application, the more complicated types (`MapMessage` and `ObjectMessage`) carry a performance cost: the expense of serializing and deserializing the data. The performance cost depends on how simple or how complicated the data is.

Message Service Factors Affecting Performance

The performance of a messaging application is affected not only by application design, but also by the message service performing the routing and delivery of messages.

The following sections discuss various message service factors that can affect performance. Understanding the effect of these factors is key to sizing a message service and diagnosing and resolving performance bottlenecks that might arise in a deployed application.

The most important factors affecting performance in a Message Queue service are the following:

- [“Hardware” on page 239](#)
- [“Operating System” on page 239](#)
- [“Java Virtual Machine \(JVM\)” on page 239](#)
- [“Connections” on page 239](#)
- [“Broker Limits and Behaviors” on page 241](#)
- [“Message Service Architecture” on page 241](#)
- [“Data Store Performance” on page 241](#)
- [“Client Runtime Configuration” on page 242](#)

The sections below describe the effect of each of these factors on messaging performance.

Hardware

For both the Message Queue broker and client applications, CPU processing speed and available memory are primary determinants of message service performance. Many software limitations can be eliminated by increasing processing power, while adding memory can increase both processing speed and capacity. However, it is generally expensive to overcome bottlenecks simply by upgrading your hardware.

Operating System

Because of the efficiencies of different operating systems, performance can vary, even assuming the same hardware platform. For example, the thread model employed by the operating system can have an important effect on the number of concurrent connections a broker can support. In general, all hardware being equal, Solaris is generally faster than Linux, which is generally faster than Windows.

Java Virtual Machine (JVM)

The broker is a Java process that runs in and is supported by the host JVM. As a result, JVM processing is an important determinant of how fast and efficiently a broker can route and deliver messages.

In particular, the JVM's management of memory resources can be critical. Sufficient memory has to be allocated to the JVM to accommodate increasing memory loads. In addition, the JVM periodically reclaims unused memory, and this memory reclamation can delay message processing. The larger the JVM memory heap, the longer the potential delay that might be experienced during memory reclamation.

Connections

The number and speed of connections between client and broker can affect the number of messages that a message service can handle as well as the speed of message delivery.

Broker Connection Limits

All access to the broker is by way of connections. Any limit on the number of concurrent connections can affect the number of producing or consuming clients that can concurrently use the broker.

The number of connections to a broker is generally limited by the number of threads available. Message Queue can be configured to support either a dedicated thread model or a shared thread model (see [“Thread Pool Management” on page 96](#)).

The dedicated thread model is very fast because each connection has dedicated threads, however the number of connections is limited by the number of threads available (one input thread and one output thread for each connection). The shared thread model places no limit on the number of connections, however there is significant overhead and throughput delays in sharing threads among a number of connections, especially when those connections are busy.

Transport Protocols

Message Queue software allows clients to communicate with the broker using various low-level transport protocols. Message Queue supports the connection services (and corresponding protocols) described in “[Configuring Connection Services](#)” on page 93.

The choice of protocols is based on application requirements (encrypted, accessible through a firewall), but the choice affects overall performance.

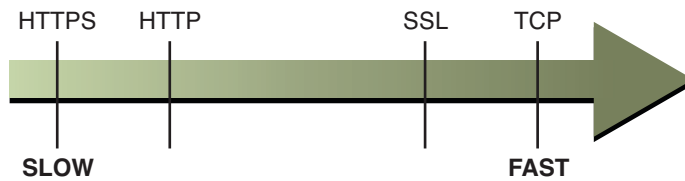


FIGURE 13-2 Transport Protocol Speeds

Our tests compared throughput for TCP and SSL for two cases: a high-reliability scenario (1k persistent messages sent to topic destinations with durable subscriptions and using `AUTO_ACKNOWLEDGE` acknowledgment mode) and a high-performance scenario (1k nonpersistent messages sent to topic destinations without durable subscriptions and using `DUPS_OK_ACKNOWLEDGE` acknowledgment mode).

In general we found that protocol has less effect in the high-reliability case. This is probably because the persistence overhead required in the high-reliability case is a more important factor in limiting throughput than the protocol speed. Additionally:

- TCP provides the fastest method to communicate with the broker.
- SSL is 50 to 70 percent slower than TCP when it comes to sending and receiving messages (50 percent for persistent messages, closer to 70 percent for nonpersistent messages). Additionally, establishing the initial connection is slower with SSL (it might take several seconds) because the client and broker (or Web Server in the case of HTTPS) need to establish a private key to be used when encrypting the data for transmission. The performance drop is caused by the additional processing required to encrypt and decrypt each low-level TCP packet.
- HTTP is slower than either the TCP or SSL. It uses a servlet that runs on a Web server as a proxy between the client and the broker. Performance overhead is involved in encapsulating packets in HTTP requests and in the requirement that messages go through two hops--client to servlet, servlet to broker--to reach the broker.
- HTTPS is slower than HTTP because of the additional overhead required to encrypt the packet between client and servlet and between servlet and broker.

Message Service Architecture

A Message Queue message service can be implemented as a single broker or as a cluster consisting of multiple interconnected broker instances.

As the number of clients connected to a broker increases, and as the number of messages being delivered increases, a broker will eventually exceed resource limitations such as file descriptor, thread, and memory limits. One way to accommodate increasing loads is to add more broker instances to a Message Queue message service, distributing client connections and message routing and delivery across multiple brokers.

In general, this scaling works best if clients are evenly distributed across the cluster, especially message producing clients. Because of the overhead involved in delivering messages between the brokers in a cluster, clusters with limited numbers of connections or limited message delivery rates, might exhibit lower performance than a single broker.

You might also use a broker cluster to optimize network bandwidth. For example, you might want to use slower, long distance network links between a set of remote brokers within a cluster, while using higher speed links for connecting clients to their respective broker instances.

For more information on clusters, see [Chapter 10, “Configuring and Managing Broker Clusters”](#)

Broker Limits and Behaviors

The message throughput that a broker might be required to handle is a function of the use patterns of the messaging applications the broker supports. However, the broker is limited in resources: memory, CPU cycles, and so forth. As a result, it would be possible for a broker to become overwhelmed to the point where it becomes unresponsive or unstable.

The Message Queue message broker has mechanisms built in for managing memory resources and preventing the broker from running out of memory. These mechanisms include configurable limits on the number of messages or message bytes that can be held by a broker or its individual physical destinations, and a set of behaviors that can be instituted when physical destination limits are reached.

With careful monitoring and tuning, these configurable mechanisms can be used to balance the inflow and outflow of messages so that system overload cannot occur. While these mechanisms consume overhead and can limit message throughput, they nevertheless maintain operational integrity.

Data Store Performance

Message Queue supports both file-based and JDBC-based persistence modules. File-based persistence uses individual files to store persistent data. JDBC-based persistence uses a Java Database Connectivity (JDBC) interface and requires a JDBC-compliant data store. File-based persistence is generally faster than JDBC-based; however, some users prefer the redundancy and administrative control provided by a JDBC-compliant store.

In the case of file-based persistence, you can maximize reliability by specifying that persistence operations synchronize the in-memory state with the data store. This helps eliminate data loss due to system crashes, but at the expense of performance.

Client Runtime Configuration

The Message Queue client runtime provides client applications with an interface to the Message Queue message service. It supports all the operations needed for clients to send messages to physical destinations and to receive messages from such destinations. The client runtime is configurable (by setting connection factory attribute values), allowing you to control aspects of its behavior, such as connection flow metering, consumer flow limits, and connection flow limits, that can improve performance and message throughput. See [“Client Runtime Message Flow Adjustments” on page 246](#) for more information on these features and the attributes used to configure them.

Adjusting Configuration To Improve Performance

The following sections explain how configuration adjustments can affect performance.

System Adjustments

The following sections describe adjustments you can make to the operating system, JVM, communication protocols, and persistent data store.

Solaris Tuning: CPU Utilization, Paging/Swapping/Disk I/O

See your system documentation for tuning your operating system.

Java Virtual Machine Adjustments

By default, the broker uses a JVM heap size of 192MB. This is often too small for significant message loads and should be increased.

When the broker gets close to exhausting the JVM heap space used by Java objects, it uses various techniques such as flow control and message swapping to free memory. Under extreme circumstances it even closes client connections in order to free the memory and reduce the message inflow. Hence it is desirable to set the maximum JVM heap space high enough to avoid such circumstances.

However, if the maximum Java heap space is set too high, in relation to system physical memory, the broker can continue to grow the Java heap space until the entire system runs out of memory. This can result in diminished performance, unpredictable broker crashes, and/or affect the behavior of other applications and services running on the system. In general, you need to allow enough physical memory for the operating system and other applications to run on the machine.

In general it is a good idea to evaluate the normal and peak system memory footprints, and configure the Java heap size so that it is large enough to provide good performance, but not so large as to risk system memory problems.

To change the minimum and maximum heap size for the broker, use the `-vmargs` command line option when starting the broker. For example:

```
/usr/bin/imqbrokerd -vmargs "-Xms256m -Xmx1024m"
```

This command will set the starting Java heap size to 256MB and the maximum Java heap size to 1GB.

- On Solaris or Linux, if starting the broker via `/etc/rc*` (that is, `/etc/init.d/imq`), specify broker command line arguments in the file `/etc/imq/imqbrokerd.conf` (Solaris) or `/etc/opt/sun/mq/imqbrokerd.conf` (Linux). See the comments in that file for more information.
- On Windows, if starting the broker as a Window's service, specify JVM arguments using the `-vmargs` option to the `imqsvcadmin install` command. See [“Service Administrator Utility” on page 297 in Chapter 15, “Command Line Reference”](#)

In any case, verify settings by checking the broker's log file or using the `imqcmd metrics bkr -m cxn` command.

Tuning Transport Protocols

Once a protocol that meets application needs has been chosen, additional tuning (based on the selected protocol) might improve performance.

A protocol's performance can be modified using the following three broker properties:

- `imq.protocol.protocolType.nodelay`
- `imq.protocol.protocolType.inbufsz`
- `imq.protocol.protocolType.outbufsz`

For TCP and SSL protocols, these properties affect the speed of message delivery between client and broker. For HTTP and HTTPS protocols, these properties affect the speed of message delivery between the Message Queue tunnel servlet (running on a Web server) and the broker. For HTTP/HTTPS protocols there are additional properties that can affect performance (see [“HTTP/HTTPS Tuning” on page 244](#)).

The protocol tuning properties are described in the following sections.

nodelay

The `nodelay` property affects Nagle's algorithm (the value of the `TCP_NODELAY` socket-level option on TCP/IP) for the given protocol. Nagle's algorithm is used to improve TCP performance on systems using slow connections such as wide-area networks (WANs).

When the algorithm is used, TCP tries to prevent several small chunks of data from being sent to the remote system (by bundling the data in larger packets). If the data written to the socket does not fill the required buffer size, the protocol delays sending the packet until either the buffer is filled or a specific delay time has elapsed. Once the buffer is full or the timeout has occurred, the packet is sent.

For most messaging applications, performance is best if there is no delay in the sending of packets (Nagle's algorithm is not enabled). This is because most interactions between client and broker are request/response interactions: the client sends a packet of data to the broker and waits for a response. For example, typical interactions include:

- Creating a connection
- Creating a producer or consumer
- Sending a persistent message (the broker confirms receipt of the message)
- Sending a client acknowledgment in an `AUTO_ACKNOWLEDGE` or `CLIENT_ACKNOWLEDGE` session (the broker confirms processing of the acknowledgment)

For these interactions, most packets are smaller than the buffer size. This means that if Nagle's algorithm is used, the broker delays several milliseconds before sending a response to the consumer.

However, Nagle's algorithm may improve performance in situations where connections are slow and broker responses are not required. This would be the case where a client sends a nonpersistent message or where a client acknowledgment is not confirmed by the broker (`DUPS_OK_ACKNOWLEDGE` session).

inbufsz/outbufsz

The `inbufsz` property sets the size of the buffer on the input stream reading data coming in from a socket. Similarly, `outbufsz` sets the buffer size of the output stream used by the broker to write data to the socket.

In general, both parameters should be set to values that are slightly larger than the average packet being received or sent. A good rule of thumb is to set these property values to the size of the average packet plus 1 kilobyte (rounded to the nearest kilobyte). For example, if the broker is receiving packets with a body size of 1 kilobyte, the overall size of the packet (message body plus header plus properties) is about 1200 bytes; an `inbufsz` of 2 kilobytes (2048 bytes) gives reasonable performance. Increasing `inbufsz` or `outbufsz` greater than that size may improve performance slightly, but increases the memory needed for each connection.

HTTP/HTTPS Tuning

In addition to the general properties discussed in the previous two sections, HTTP/HTTPS performance is limited by how fast a client can make HTTP requests to the Web server hosting the Message Queue tunnel servlet.

A Web server might need to be optimized to handle multiple requests on a single socket. With JDK version 1.4 and later, HTTP connections to a Web server are kept alive (the socket to the Web server remains open) to minimize resources used by the Web server when it processes multiple HTTP requests. If the performance of a client application using JDK version 1.4 is slower than the same application running with an earlier JDK release, you might need to tune the Web server keep-alive configuration parameters to improve performance.

In addition to such Web server tuning, you can also adjust how often a client polls the Web server. HTTP is a request-based protocol. This means that clients using an HTTP-based protocol periodically need to check the Web server to see if messages are waiting. The `imq.http.jms.http.pullPeriod` broker property (and the corresponding `imq.https.jms.https.pullPeriod` property) specifies how often the Message Queue client runtime polls the Web server.

If the `pullPeriod` value is `-1` (the default value), the client runtime polls the server as soon as the previous request returns, maximizing the performance of the individual client. As a result, each client connection monopolizes a request thread in the Web server, possibly straining Web server resources.

If the `pullPeriod` value is a positive number, the client runtime periodically sends requests to the Web server to see if there is pending data. In this case, the client does not monopolize a request thread in the Web server. Hence, if large numbers of clients are using the Web server, you might conserve Web server resources by setting the `pullPeriod` to a positive value.

Tuning the File-based Persistent Store

For information on tuning the file-based persistent store, see [“Introduction to Persistence Services” on page 125](#).

Broker Memory Management Adjustments

You can improve performance and increase broker stability under load by properly managing broker memory. Memory management can be configured on a destination-by-destination basis or on a system-wide level (for all destinations, collectively).

Using Physical Destination Limits

To configure physical destination limits, see the properties described in [“Physical Destination Properties” on page 333](#).

Using System-Wide Limits

If message producers tend to overrun message consumers, messages can accumulate in the broker. The broker contains a mechanism for throttling back producers and swapping messages out of active memory under low memory conditions, but it is wise to set a hard limit on the total number of messages (and message bytes) that the broker can hold.

Control these limits by setting the `imq.system.max_count` and the `imq.system.max_size` broker properties.

For example:

```
imq.system.max_count=5000
```

The defined value above means that the broker will only hold up to 5000 undelivered and/or unacknowledged messages. If additional messages are sent, they are rejected by the broker. If a message is persistent then the client runtime will throw an exception when the producer tries to send the message. If the message is non-persistent, the broker silently drops the message.

When an exception is thrown in sending a message, the client should process the exception by pausing for a moment and retrying the send again. (Note that the exception will never be due to the broker's failure to receive a message; the exception is thrown by the client runtime before the message is sent to the broker.)

Client Runtime Message Flow Adjustments

This section discusses client runtime flow control behaviors that affect performance. These behaviors are configured as attributes of connection factory administered objects. For information on setting connection factory attributes, see [Chapter 11, “Managing Administered Objects”](#)

Message Flow Metering

Messages sent and received by clients (*payload messages*), as well as Message Queue control messages, pass over the same client-broker connection. Delays in the delivery of control messages, such as broker acknowledgments, can result if control messages are held up by the delivery of payload messages. To prevent this type of congestion, Message Queue meters the flow of payload messages across a connection.

Payload messages are batched (as specified with the connection factory attribute `imqConnectionFlowCount`) so that only a set number are delivered. After the batch has been delivered, delivery of payload messages is suspended and only pending control messages are delivered. This cycle repeats, as additional batches of payload messages are delivered followed by pending control messages.

The value of `imqConnectionFlowCount` should be kept low if the client is doing operations that require many responses from the broker: for example, if the client is using `CLIENT_ACKNOWLEDGE` or `AUTO_ACKNOWLEDGE` mode, persistent messages, transactions, or queue browsers, or is adding or removing consumers. If, on the other hand, the client has only simple consumers on a connection using `DUPS_OK_ACKNOWLEDGE` mode, you can increase `imqConnectionFlowCount` without compromising performance.

Message Flow Limits

There is a limit to the number of payload messages that the Message Queue client runtime can handle before encountering local resource limitations, such as memory. When this limit is approached, performance suffers. Hence, Message Queue lets you limit the number of messages per consumer (or messages per connection) that can be delivered over a connection and buffered in the client runtime, waiting to be consumed.

Consumer Flow Limits

When the number of payload messages delivered to the client runtime exceeds the value of `imqConsumerFlowLimit` for any consumer, message delivery for that consumer stops. It is resumed only when the number of unconsumed messages for that consumer drops below the value set with `imqConsumerFlowThreshold`.

The following example illustrates the use of these limits: consider the default settings for topic consumers:

```
imqConsumerFlowLimit=1000
imqConsumerFlowThreshold=50
```

When the consumer is created, the broker delivers an initial batch of 1000 messages (providing they exist) to this consumer without pausing. After sending 1000 messages, the broker stops delivery until the client runtime asks for more messages. The client runtime holds these messages until the application processes them. The client runtime then allows the application to consume at least 50% (`imqConsumerFlowThreshold`) of the message buffer capacity (i.e. 500 messages) before asking the broker to send the next batch.

In the same situation, if the threshold were 10%, the client runtime would wait for the application to consume at least 900 messages before asking for the next batch.

The next batch size is calculated as follows:

$$\text{imqConsumerFlowLimit} - (\text{current number of pending msgs in buffer})$$

So if `imqConsumerFlowThreshold` is 50%, the next batch size can fluctuate between 500 and 1000, depending on how fast the application can process the messages.

If the `imqConsumerFlowThreshold` is set too high (close to 100%), the broker will tend to send smaller batches, which can lower message throughput. If the value is set too low (close to 0%), the client may be able to finish processing the remaining buffered messages before the broker delivers the next set, again degrading message throughput. Generally speaking, unless you have specific performance or reliability concerns, you will not need to change the default value of `imqConsumerFlowThreshold` attribute.

The consumer-based flow controls (in particular, `imqConsumerFlowLimit`) are the best way to manage memory in the client runtime. Generally, depending on the client application, you

know the number of consumers you need to support on any connection, the size of the messages, and the total amount of memory that is available to the client runtime.

Connection Flow Limits

In the case of some client applications, however, the number of consumers may be indeterminate, depending on choices made by end users. In those cases, you can still manage memory using connection-level flow limits.

Connection-level flow controls limit the total number of messages buffered for *all* consumers on a connection. If this number exceeds the value of `imqConnectionFactoryLimit`, delivery of messages through the connection stops until that total drops below the connection limit. (The `imqConnectionFactoryLimit` attribute is enabled only if you set `imqConnectionFactoryLimitEnabled` to `true`.)

The number of messages queued up in a session is a function of the number of message consumers using the session and the message load for each consumer. If a client is exhibiting delays in producing or consuming messages, you can normally improve performance by redesigning the application to distribute message producers and consumers among a larger number of sessions or to distribute sessions among a larger number of connections.

Adjusting Multiple-Consumer Queue Delivery

The efficiency with which multiple queue consumers process messages in a queue destination depends on a number of factors. To achieve optimal message throughput there must be a sufficient number of consumers to keep up with the rate of message production for the queue, and the messages in the queue must be routed and then delivered to the active consumers in such a way as to maximize their rate of consumption.

The message delivery mechanism for multiple-consumer queues is that messages are delivered to consumers in batches as each consumer is ready to receive a new batch. The readiness of a consumer to receive a batch of messages depends upon configurable client runtime properties, such as `imqConsumerFlowLimit` and `imqConsumerFlowThreshold`, as described in [“Message Flow Limits” on page 247](#). As new consumers are added to a queue, they are sent a batch of messages to consume, and receive subsequent batches as they become ready.

Note – The message delivery mechanism for multiple-consumer queues described above can result in messages being consumed in an order different from the order in which they are produced.

If messages are accumulating in the queue, it is possible that there is an insufficient number of consumers to handle the message load. It is also possible that messages are being delivered to consumers in batch sizes that cause messages to be backing up on the consumers. For example,

if the batch size (`consumerFlowLimit`) is too large, one consumer might receive all the messages in a queue while other consumers receive none. If consumers are very fast, this might not be a problem. However, if consumers are relatively slow, you want messages to be distributed to them evenly, and therefore you want the batch size to be small. Although smaller batch sizes require more overhead to deliver messages to consumers, for slow consumers there is generally a net performance gain in using small batch sizes. The value of `consumerFlowLimit` can be set on a destination as well as on the client runtime: the smaller value overrides the larger one.

Troubleshooting

This chapter explains how to understand and resolve the following problems:

- “A Client Cannot Establish a Connection” on page 251
- “Connection Throughput Is Too Slow” on page 256
- “A Client Cannot Create a Message Producer” on page 257
- “Message Production Is Delayed or Slowed” on page 258
- “Messages Are Backlogged” on page 261
- “Broker Throughput Is Sporadic” on page 264
- “Messages Are Not Reaching Consumers” on page 266
- “Dead Message Queue Contains Messages” on page 267

When problems occur, it is useful to check the version number of the installed Message Queue software. Use the version number to ensure that you are using documentation whose version matches the software version. You also need the version number to report a problem to Sun. To check the version number, issue the following command:

```
imqcmd -v
```

A Client Cannot Establish a Connection

Symptoms:

- Client cannot make a new connection.
- Client cannot auto-reconnect on failed connection.

Possible causes:

- Client applications are not closing connections, causing the number of connections to exceed resource limitations.
- Broker is not running or there is a network connectivity problem.
- Connection service is inactive or paused.

- Too few threads available for the number of connections required.
- Too few file descriptors for the number of connections required on the Solaris or Linux operating system.
- TCP backlog limits the number of simultaneous new connection requests that can be established.
- Operating system limits the number of concurrent connections.
- Authentication or authorization of the user is failing.

Possible cause: Client applications are not closing connections, causing the number of connections to exceed resource limitations.

To confirm this cause of the problem: List all connections to a broker:

```
imqcmd list cxn
```

The output will list all connections and the host from which each connection has been made, revealing an unusual number of open connections for specific clients.

To resolve the problem: Rewrite the offending clients to close unused connections.

Possible cause: Broker is not running or there is a network connectivity problem.

To confirm this cause of the problem:

- Telnet to the broker's primary port (for example, the default of 7676) and verify that the broker responds with Port Mapper output.
- Verify that the broker process is running on the host.

To resolve the problem:

- Start up the broker.
- Fix the network connectivity problem.

Possible cause: Connection service is inactive or paused.

To confirm this cause of the problem: Check the status of all connection services:

```
imqcmd list svc
```

If the status of a connection service is shown as unknown or paused, clients will not be able to establish a connection using that service.

To resolve the problem:

- If the status of a connection service is shown as unknown, it is missing from the active service list (`imq.service.active`). In the case of SSL-based services, the service might also be improperly configured, causing the broker to make the following entry in the broker log:

```
ERROR [B3009]: Unable to start service ssljms:  
[B4001]: Unable to open protocol tls for ssljms service...
```

followed by an explanation of the underlying cause of the exception.

To properly configure SSL services, see [“Message Encryption” on page 159](#).

- If the status of a connection service is shown as paused, resume the service (see [“Pausing and Resuming a Connection Service” on page 97](#)).

Possible cause: Too few threads available for the number of connections required.

To confirm this cause of the problem: Check for the following entry in the broker log:

```
WARNING [B3004]: No threads are available to process a new connection on service
...
Closing the new connection.
```

Also check the number of connections on the connection service and the number of threads currently in use, using one of the following formats:

```
imqcmd query svc -n serviceName
imqcmd metrics svc -n serviceName -m cxn
```

Each connection requires two threads: one for incoming messages and one for outgoing messages (see [“Thread Pool Management” on page 96](#)).

To resolve the problem:

- If you are using a dedicated thread pool model (`imq.serviceName.threadpool_model=dedicated`), the maximum number of connections is half the maximum number of threads in the thread pool. Therefore, to increase the number of connections, increase the size of the thread pool (`imq.serviceName.max_threads`) or switch to the shared thread pool model.
- If you are using a shared thread pool model (`imq.serviceName.threadpool_model=shared`), the maximum number of connections is half the product of the connection monitor limit (`imq.serviceName.connectionMonitor_limit`) and the maximum number of threads (`imq.serviceName.max_threads`). Therefore, to increase the number of connections, increase the size of the thread pool or increase the connection monitor limit.
- Ultimately, the number of supportable connections (or the throughput on connections) will reach input/output limits. In such cases, use a multiple-broker cluster to distribute connections among the broker instances within the cluster.

Possible cause: Too few file descriptors for the number of connections required on the Solaris or Linux platform.

For more information about this issue, see [“Setting the File Descriptor Limit” on page 68](#).

To confirm this cause of the problem: Check for an entry in the broker log similar to the following:

```
Too many open files
```

To resolve the problem: Increase the file descriptor limit, as described in the man page for the `ulimit` command.

Possible cause: TCP backlog limits the number of simultaneous new connection requests that can be established.

The TCP backlog places a limit on the number of simultaneous connection requests that can be stored in the system backlog (`imq.portmapper.backlog`) before the Port Mapper rejects additional requests. (On the Windows platform there is a hard-coded backlog limit of 5 for Windows desktops and 200 for Windows servers.)

The rejection of requests because of backlog limits is usually a transient phenomenon, due to an unusually high number of simultaneous connection requests.

To confirm this cause of the problem: Examine the broker log. First, check to see whether the broker is accepting some connections during the same time period that it is rejecting others. Next, check for messages that explain rejected connections. If you find such messages, the TCP backlog is probably not the problem, because the broker does not log connection rejections due to the TCP backlog. If some successful connections are logged, and no connection rejections are logged, the TCP backlog is probably the problem.

To resolve the problem:

- Program the client to retry the attempted connection after a short interval of time (this normally works because of the transient nature of this problem).
- Increase the value of `imq.portmapper.backlog`.
- Check that clients are not closing and then opening connections too often.

Possible cause: Operating system limits the number of concurrent connections.

The Windows operating system license places limits on the number of concurrent remote connections that are supported.

To confirm this cause of the problem: Check that there are plenty of threads available for connections (using `imqcmd query svc`) and check the terms of your Windows license agreement. If you can make connections from a local client, but not from a remote client, operating system limitations might be the cause of the problem.

To resolve the problem:

- Upgrade the Windows license to allow more connections.
- Distribute connections among a number of broker instances by setting up a multiple-broker cluster.

Possible cause: Authentication or authorization of the user is failing.

The authentication may be failing for any of the following reasons:

- Incorrect password
- No entry for user in user repository
- User does not have access permission for connection service

To confirm this cause of the problem: Check entries in the broker log for the Forbidden error message. This will indicate an authentication error, but will not indicate the reason for it.

- If you are using a file-based user repository, enter the following command:
`imqusermgr list -i instanceName -u userName`
 If the output shows a user, the wrong password was probably submitted. If the output shows the following error, there is no entry for the user in the user repository:
 Error [B3048]: User does not exist in the password file
- If you are using an LDAP server user repository, use the appropriate tools to check whether there is an entry for the user.
- Check the access control file to see whether there are restrictions on access to the connection service.

To resolve the problem:

- If the wrong password was used, provide the correct password.
- If there is no entry for the user in the user repository, add one (see [“Adding a User to the Repository” on page 142](#)).
- If the user does not have access permission for the connection service, edit the access control file to grant such permission (see [“Authorization Rules for Connection Services” on page 157](#)).

Possible cause: Authentication or authorization of the user is failing.

Authentication may be failing for any of the following reasons:

- No entry for user in user repository
- Incorrect password
- User does not have access permission for connection service

To confirm this cause of the problem

1. Check entries in the broker log for the error message Forbidden. This will indicate an authentication error, but will not indicate the reason for it.
2. Check the user repository for an entry for this user:

- If you are using a flat-file user repository, enter the command

```
imqusermgr list -i instanceName -u userName
```

If the output shows the error

```
Error [B3048]: User does not exist in the password file
```

then there is no entry for the user in the user repository:

- If you are using an LDAP user repository, use the appropriate tools to check whether there is an entry for the user.
3. If the output from step 2 does show a user entry, the wrong password was probably provided.

4. Check the access control file to see whether there are restrictions on access to the connection service.

To resolve the problem

- If there is no entry for the user in the user repository, add one (see [“Adding a User to the Repository” on page 142](#)).
- If the wrong password was used, provide the correct password.
- If the user does not have access permission for the connection service, edit the access control file to grant such permission (see [“Authorization Rules for Connection Services” on page 157](#)).

Connection Throughput Is Too Slow

Symptoms:

- Message throughput does not meet expectations.
- Message input/output rates are not limited by an insufficient number of supported connections (as described in [“A Client Cannot Establish a Connection” on page 251](#)).

Possible causes:

- [Network connection or WAN is too slow.](#)
- [Connection service protocol is inherently slow compared to TCP.](#)
- [Connection service protocol is not optimally tuned.](#)
- [Messages are so large that they consume too much bandwidth.](#)
- [What appears to be slow connection throughput is actually a bottleneck in some other step of the message delivery process.](#)

Possible cause: [Network connection or WAN is too slow.](#)

To confirm this cause of the problem:

- Ping the network, to see how long it takes for the ping to return, and consult a network administrator.
- Send and receive messages using local clients and compare the delivery time with that of remote clients (which use a network link).

To resolve the problem: Upgrade the network link.

Possible cause: [Connection service protocol is inherently slow compared to TCP.](#)

For example, SSL-based or HTTP-based protocols are slower than TCP (see [“Transport Protocols” on page 240](#)).

To confirm this cause of the problem: If you are using SSL-based or HTTP-based protocols, try using TCP and compare the delivery times.

To resolve the problem: Application requirements usually dictate the protocols being used, so there is little you can do other than attempt to tune the protocol as described in [“Tuning Transport Protocols” on page 243](#).

Possible cause: Connection service protocol is not optimally tuned.

To confirm this cause of the problem: Try tuning the protocol to see whether it makes a difference.

To resolve the problem: Try tuning the protocol, as described in [“Tuning Transport Protocols” on page 243](#).

Possible cause: Messages are so large that they consume too much bandwidth.

To confirm this cause of the problem: Try running your benchmark with smaller-sized messages.

To resolve the problem:

- Have application developers modify the application to use the message compression feature, which is described in the *Message Queue Developer's Guide for Java Clients*.
- Use messages as notifications of data to be sent, but move the data using another protocol.

Possible cause: What appears to be slow connection throughput is actually a bottleneck in some other step of the message delivery process.

To confirm this cause of the problem: If what appears to be slow connection throughput cannot be explained by any of the causes above, see [“Factors Affecting Performance” on page 232](#) for other possible bottlenecks and check for symptoms associated with the following problems:

- [“Message Production Is Delayed or Slowed” on page 258](#)
- [“Messages Are Backlogged” on page 261](#)
- [“Broker Throughput Is Sporadic” on page 264](#)

To resolve the problem: Follow the problem resolution guidelines provided in the troubleshooting sections listed above.

A Client Cannot Create a Message Producer

Symptom:

- A message producer cannot be created for a physical destination; the client receives an exception.

Possible causes:

- [A physical destination has been configured to allow only a limited number of producers.](#)
- [The user is not authorized to create a message producer due to settings in the access control file.](#)

Possible cause: A physical destination has been configured to allow only a limited number of producers.

One of the ways of avoiding the accumulation of messages on a physical destination is to limit the number of producers (`maxNumProducers`) that it supports.

To confirm this cause of the problem: Check the physical destination:

```
imqcmd query dst
```

(see [“Viewing Physical Destination Information” on page 112](#)). The output will show the current number of producers and the value of `maxNumProducers`. If the two values are the same, the number of producers has reached its configured limit. When a new producer is rejected by the broker, the broker returns the exception

```
ResourceAllocationException [C4088]: A JMS destination limit was reached
and makes the following entry in the broker log:
```

```
[B4183]: Producer can not be added to destination
```

To resolve the problem: Increase the value of the `maxNumProducers` property (see [“Updating Physical Destination Properties” on page 112](#)).

Possible cause: The user is not authorized to create a message producer due to settings in the access control file.

To confirm this cause of the problem: When a new producer is rejected by the broker, the broker returns the exception

```
JMSSecurityException [C4076]: Client does not have permission to create producer
on destination
```

and makes the following entries in the broker log:

```
[B2041]: Producer on destination denied
```

```
[B4051]: Forbidden guest.
```

To resolve the problem: Change the access control properties to allow the user to produce messages (see [“Authorization Rules for Physical Destinations” on page 158](#)).

Message Production Is Delayed or Slowed

Symptoms:

- When sending persistent messages, the `send` method does not return and the client blocks.
- When sending a persistent message, the client receives an exception.
- A producing client slows down.

Possible causes:

- The broker is backlogged and has responded by slowing message producers.
- The broker cannot save a persistent message to the data store.
- Broker acknowledgment timeout is too short.

- [A producing client is encountering JVM limitations.](#)

Possible cause: The broker is backlogged and has responded by slowing message producers.

A backlogged broker accumulates messages in broker memory. When the number of messages or message bytes in physical destination memory reaches configured limits, the broker attempts to conserve memory resources in accordance with the specified limit behavior. The following limit behaviors slow down message producers:

- **FLOW_CONTROL:** The broker does not immediately acknowledge receipt of persistent messages (thereby blocking a producing client).
- **REJECT_NEWEST:** The broker rejects new persistent messages.

Similarly, when the number of messages or message bytes in brokerwide memory (for all physical destinations) reaches configured limits, the broker will attempt to conserve memory resources by rejecting the newest messages. Also, when system memory limits are reached because physical destination or brokerwide limits have not been set properly, the broker takes increasingly serious action to prevent memory overload. These actions include throttling back message producers.

To confirm this cause of the problem: When a message is rejected by the broker because of configured message limits, the broker returns the exception

`JMSEException [C4036]: A server error occurred`

and makes the following entry in the broker log:

`[B2011]: Storing of JMS message from IMQconn failed`

This message is followed by another indicating the limit that has been reached:

`[B4120]: Cannot store message on destination destName because capacity of maxNumMsgs would be exceeded.`

if the exceeded message limit is on a physical destination, or

`[B4024]: The maximum number of messages currently in the system has been exceeded, rejecting message.`

if the limit is brokerwide.

More generally, you can check for message limit conditions before the rejections occur as follows:

- Query physical destinations and the broker and inspect their configured message limit settings.
- Monitor the number of messages or message bytes currently in a physical destination or in the broker as a whole, using the appropriate `imqcmd` commands. See [Chapter 20, “Metrics Information Reference,”](#) for information about metrics you can monitor and the commands you use to obtain them.

To resolve the problem:

- Modify the message limits on a physical destination (or brokerwide), being careful not to exceed memory resources.

In general, you should manage memory at the individual destination level, so that brokerwide message limits are never reached. For more information, see [“Broker Memory Management Adjustments” on page 245](#).

- Change the limit behaviors on a destination so as not to slow message production when message limits are reached, but rather to discard messages in memory.

For example, you can specify the `REMOVE_OLDEST` and `REMOVE_LOW_PRIORITY` limit behaviors, which delete messages that accumulate in memory (see [Table 17-1](#)).

Possible cause: The broker cannot save a persistent message to the data store.

If the broker cannot access a data store or write a persistent message to it, the producing client is blocked. This condition can also occur if destination or brokerwide message limits are reached, as described above.

To confirm this cause of the problem: If the broker is unable to write to the data store, it makes one of the following entries in the broker log:

```
[B2011]: Storing of JMS message from connectionID failed
[B4004]: Failed to persist message messageID
```

To resolve the problem:

- In the case of file-based persistence, try increasing the disk space of the file-based data store.
- In the case of a JDBC-compliant data store, check that JDBC-based persistence is properly configured (see [“Configuring a JDBC-Based Data Store” on page 130](#)). If so, consult your database administrator to troubleshoot other database problems.

Possible cause: Broker acknowledgment timeout is too short.

Because of slow connections or a lethargic broker (caused by high CPU utilization or scarce memory resources), a broker may require more time to acknowledge receipt of a persistent message than allowed by the value of the connection factory’s `imqAckTimeout` attribute.

To confirm this cause of the problem: If the `imqAckTimeout` value is exceeded, the broker returns the exception

```
JMSException [C4000]: Packet acknowledge failed
```

To resolve the problem: Change the value of the `imqAckTimeout` connection factory attribute (see [“Reliability And Flow Control” on page 197](#)).

Possible cause: A producing client is encountering JVM limitations.

To confirm this cause of the problem:

- Find out whether the client application receives an out-of-memory error.
- Check the free memory available in the JVM heap, using runtime methods such as `freeMemory`, `maxMemory`, and `totalMemory`.

To resolve the problem: Adjust the JVM (see [“Java Virtual Machine Adjustments” on page 242](#)).

Messages Are Backlogged

Symptoms:

- Message production is delayed or produced messages are rejected by the broker.
- Messages take an unusually long time to reach consumers.
- The number of messages or message bytes in the broker (or in specific destinations) increases steadily over time.

To see whether messages are accumulating, check how the number of messages or message bytes in the broker changes over time and compare to configured limits. First check the configured limits:

```
imqcmd query bkr
```

Note – The `imqcmd metrics bkr` subcommand does not display this information.

Then check for message accumulation in each destination:

```
imqcmd list dst
```

To see whether messages have exceeded configured destination or brokerwide limits, check the broker log for the entry

```
[B2011]: Storing of JMS message from ... failed.
```

This entry will be followed by another identifying the limit that has been exceeded.

Possible causes:

- There are inactive durable subscriptions on a topic destination.
- Too few consumers are available to consume messages in a queue.
- Message consumers are processing too slowly to keep up with message producers.
- Client acknowledgment processing is slowing down message consumption.
- The broker cannot keep up with produced messages.
- Client code defects; consumers are not acknowledging messages.

Possible cause: There are inactive durable subscriptions on a topic destination.

If a durable subscription is inactive, messages are stored in a destination until the corresponding consumer becomes active and can consume the messages.

To confirm this cause of the problem: Check the state of durable subscriptions on each topic destination:

```
imqcmd list dur -d destName
```

To resolve the problem:

- Purge all messages for the offending durable subscriptions (see [“Managing Durable Subscriptions” on page 121](#)).
- Specify message limit and limit behavior attributes for the topic (see [Table 17–1](#)). For example, you can specify the `REMOVE_OLDEST` and `REMOVE_LOW_PRIORITY` limit behaviors, which delete messages that accumulate in memory.
- Purge all messages from the corresponding destinations (see [“Purging a Physical Destination” on page 111](#)).
- Limit the time messages can remain in memory by rewriting the producing client to set a time-to-live value on each message. You can override any such settings for all producers sharing a connection by setting the `imqOverrideJMSEExpiration` and `imqJMSEExpiration` connection factory attributes (see [“Message Header Overrides” on page 344](#)).

Possible cause: Too few consumers are available to consume messages in a multiple-consumer queue.

If there are too few active consumers to which messages can be delivered, a queue destination can become backlogged as messages accumulate. This condition can occur for any of the following reasons:

- Too few active consumers exist for the destination.
- Consuming clients have failed to establish connections.
- No active consumers use a selector that matches messages in the queue.

To confirm this cause of the problem: To help determine the reason for unavailable consumers, check the number of active consumers on a destination:

```
imqcmd metrics dst -n destName -t q -m con
```

To resolve the problem: Depending on the reason for unavailable consumers,

- Create more active consumers for the queue by starting up additional consuming clients.
- Adjust the `imq.consumerFlowLimit` broker property to optimize queue delivery to multiple consumers (see [“Adjusting Multiple-Consumer Queue Delivery” on page 248](#)).
- Specify message limit and limit behavior attributes for the queue (see [Table 17–1](#)). For example, you can specify the `REMOVE_OLDEST` and `REMOVE_LOW_PRIORITY` limit behaviors, which delete messages that accumulate in memory.
- Purge all messages from the corresponding destinations (see [“Purging a Physical Destination” on page 111](#)).
- Limit the time messages can remain in memory by rewriting the producing client to set a time-to-live value on each message. You can override any such setting for all producers sharing a connection by setting the `imqOverrideJMSEExpiration` and `imqJMSEExpiration` connection factory attributes (see [“Message Header Overrides” on page 344](#)).

Possible cause: Message consumers are processing too slowly to keep up with message producers.

In this case, topic subscribers or queue receivers are consuming messages more slowly than the producers are sending messages. One or more destinations are getting backlogged with messages because of this imbalance.

To confirm this cause of the problem: Check for the rate of flow of messages into and out of the broker:

```
imqcmd metrics bkr -m rts
```

Then check flow rates for each of the individual destinations:

```
imqcmd metrics bkr -t destType -n destName -m rts
```

To resolve the problem:

- Optimize consuming client code.
- For queue destinations, increase the number of active consumers (see [“Adjusting Multiple-Consumer Queue Delivery”](#) on page 248).

Possible cause: Client acknowledgment processing is slowing down message consumption.

Two factors affect the processing of client acknowledgments:

- Significant broker resources can be consumed in processing client acknowledgments. As a result, message consumption may be slowed in those acknowledgment modes in which consuming clients block until the broker confirms client acknowledgments.
- JMS payload messages and Message Queue control messages (such as client acknowledgments) share the same connection. As a result, control messages can be held up by JMS payload messages, slowing message consumption.

To confirm this cause of the problem:

- Check the flow of messages relative to the flow of packets. If the number of packets per second is out of proportion to the number of messages, client acknowledgments may be a problem.

- Check to see whether the client has received the following exception:

```
JMSEException [C4000]: Packet acknowledge failed
```

To resolve the problem:

- Modify the acknowledgment mode used by clients: for example, switch to DUPS_OK_ACKNOWLEDGE or CLIENT_ACKNOWLEDGE.
- If using CLIENT_ACKNOWLEDGE or transacted sessions, group a larger number of messages into a single acknowledgment.
- Adjust consumer and connection flow control parameters (see [“Client Runtime Message Flow Adjustments”](#) on page 246).

Possible cause: The broker cannot keep up with produced messages.

In this case, messages are flowing into the broker faster than the broker can route and dispatch them to consumers. The sluggishness of the broker can be due to limitations in any or all of the following:

- CPU
- Network socket read/write operations

- Disk read/write operations
- Memory paging
- Persistent store
- JVM memory limits

To confirm this cause of the problem: Check that none of the other possible causes of this problem are responsible.

To resolve the problem:

- Upgrade the speed of your computer or data store.
- Use a broker cluster to distribute the load among multiple broker instances.

Possible cause: Client code defects; consumers are not acknowledging messages.

Messages are held in a destination until they have been acknowledged by all consumers to which they have been sent. If a client is not acknowledging consumed messages, the messages accumulate in the destination without being deleted.

For example, client code might have the following defects:

- Consumers using the `CLIENT_ACKNOWLEDGE` acknowledgment mode or transacted session may not be calling `Session.acknowledge` or `Session.commit` regularly.
- Consumers using the `AUTO_ACKNOWLEDGE` acknowledgment mode may be hanging for some reason.

To confirm this cause of the problem: First check all other possible causes listed in this section. Next, list the destination with the following command:

```
imqcmd list dst
```

Notice whether the number of messages listed under the `UnAcked` header is the same as the number of messages in the destination. Messages under this header were sent to consumers but not acknowledged. If this number is the same as the total number of messages, then the broker has sent all the messages and is waiting for acknowledgment.

To resolve the problem: Request the help of application developers in debugging this problem.

Broker Throughput Is Sporadic

Symptom:

- Message throughput sporadically drops and then resumes normal performance.

Possible causes:

- The broker is very low on memory resources.
- JVM memory reclamation (garbage collection) is taking place.
- The JVM is using the just-in-time compiler to speed up performance.

Possible cause: The broker is very low on memory resources.

Because destination and broker limits were not properly set, the broker takes increasingly serious action to prevent memory overload; this can cause the broker to become sluggish until the message backlog is cleared.

To confirm this cause of the problem: Check the broker log for a low memory condition [B1089]: In low memory condition, broker is attempting to free up resources followed by an entry describing the new memory state and the amount of total memory being used. Also check the free memory available in the JVM heap:

```
imqcmd metrics bkr -m cxn
```

Free memory is low when the value of total JVM memory is close to the maximum JVM memory value.

To resolve the problem:

- Adjust the JVM (see [“Java Virtual Machine Adjustments”](#) on page 242).
- Increase system swap space.

Possible cause: JVM memory reclamation (garbage collection) is taking place.

Memory reclamation periodically sweeps through the system to free up memory. When this occurs, all threads are blocked. The larger the amount of memory to be freed up and the larger the JVM heap size, the longer the delay due to memory reclamation.

To confirm this cause of the problem: Monitor CPU usage on your computer. CPU usage drops when memory reclamation is taking place.

Also start your broker using the following command line options:

```
-vmargs -verbose:gc
```

Standard output indicates the time when memory reclamation takes place.

To resolve the problem: In computers with multiple CPUs, set the memory reclamation to take place in parallel:

```
-XX:+UseParallelGC=true
```

Possible cause: The JVM is using the just-in-time compiler to speed up performance.

To confirm this cause of the problem: Check that none of the other possible causes of this problem are responsible.

To resolve the problem: Let the system run for awhile; performance should improve.

Messages Are Not Reaching Consumers

Symptom:

- Messages sent by producers are not received by consumers.

Possible causes:

- Limit behaviors are causing messages to be deleted on the broker.
- Message timeout value is expiring.
- Clocks are not synchronized.
- Consuming client failed to start message delivery on a connection.

Possible cause: Limit behaviors are causing messages to be deleted on the broker.

When the number of messages or message bytes in destination memory reach configured limits, the broker attempts to conserve memory resources. Three of the configurable behaviors adopted by the broker when these limits are reached will cause messages to be lost:

- REMOVE_OLDEST: Delete the oldest messages.
- REMOVE_LOW_PRIORITY: Delete the lowest-priority messages according to age.
- REJECT_NEWEST: Reject new persistent messages.

To confirm this cause of the problem: Use the QBrowser demo application to inspect the contents of the dead message queue (see [“To Inspect the Dead Message Queue” on page 272](#)).

Check whether the JMS_SUN_DMQ_UNDELIVERED_REASON property of messages in the queue has the value REMOVE_OLDEST or REMOVE_LOW_PRIORITY.

To resolve the problem: Increase the destination limits. For example:

```
imqcmd update dst -n MyDest -o maxNumMsgs=1000
```

Possible cause: Message timeout value is expiring.

The broker deletes messages whose timeout value has expired. If a destination gets sufficiently backlogged with messages, messages whose time-to-live value is too short might be deleted.

To confirm this cause of the problem: Use the QBrowser demo application to inspect the contents of the dead message queue (see [“To Inspect the Dead Message Queue” on page 272](#)).

Check whether the JMS_SUN_DMQ_UNDELIVERED_REASON property of messages in the queue has the value EXPIRED.

To resolve the problem: Contact the application developers and have them increase the time-to-live value.

Possible cause: The broker clock and producer clock are not synchronized.

If clocks are not synchronized, broker calculations of message lifetimes can be wrong, causing messages to exceed their expiration times and be deleted.

To confirm this cause of the problem: Use the QBrowser demo application to inspect the contents of the dead message queue (see [“To Inspect the Dead Message Queue” on page 272](#)).

Check whether the `JMS_SUN_DMQ_UNDELIVERED_REASON` property of messages in the queue has the value `EXPIRED`.

In the broker log file, look for any of the following messages: B2102, B2103, B2104. These messages all report that possible clock skew was detected.

To resolve the problem: Check that you are running a time synchronization program, as described in [“Preparing System Resources” on page 67](#).

Possible cause: Consuming client failed to start message delivery on a connection.

Messages cannot be delivered until client code establishes a connection and starts message delivery on the connection.

To confirm this cause of the problem: Check that client code establishes a connection and starts message delivery.

To resolve the problem: Rewrite the client code to establish a connection and start message delivery.

Dead Message Queue Contains Messages

Symptom:

- When you list destinations, you see that the dead message queue contains messages. For example, issue a command like the following:

```
imqcmd list dst
```

After you supply a user name and password, output like the following appears:

Listing all the destinations on the broker specified by:

```
-----
Host          Primary Port
-----
localhost     7676
-----
```

Name	Type	State	Producers	Consumers	Msgs		
				Total	Count	UnAck	Avg Size
MyDest	Queue	RUNNING	0	0	5	0	1177.0
mq.sys.dmq	Queue	RUNNING	0	0	35	0	1422.0

Successfully listed destinations.

In this example, the dead message queue, `mq.sys.dmq`, contains 35 messages.

Possible causes:

- The number of messages, or their sizes, exceed destination limits.
- The broker clock and producer clock are not synchronized.

- An unexpected broker error has occurred.
- Consumers are not receiving messages before they time out.

There are a number of possible reasons for messages to time out:

- There are too many producers for the number of consumers.
- Producers are faster than consumers.
- A consumer is too slow.
- Clients are not committing messages.
- Consumers are failing to acknowledge messages.
- Durable consumers are inactive.

Possible cause: The number of messages, or their sizes, exceed destination limits.

To confirm this cause of the problem: Use the QBrowser demo application to inspect the contents of the dead message queue (see [“To Inspect the Dead Message Queue” on page 272](#)).

Check the values for the following message properties:

- JMS_SUN_DMQ_UNDELIVERED_REASON
- JMS_SUN_DMQ_UNDELIVERED_COMMENT
- JMS_SUN_DMQ_UNDELIVERED_TIMESTAMP

Under JMS Headers, scroll down to the value for JMSDestination to determine the destination whose messages are becoming dead.

To resolve the problem: Increase the destination limits. For example:

```
imqcmd update dst -n MyDest -o maxNumMsgs=1000
```

Possible cause: The broker clock and producer clock are not synchronized.

If clocks are not synchronized, broker calculations of message lifetimes can be wrong, causing messages to exceed their expiration times and be deleted.

To confirm this cause of the problem: Use the QBrowser demo application to inspect the contents of the dead message queue (see [“To Inspect the Dead Message Queue” on page 272](#)).

Check whether the JMS_SUN_DMQ_UNDELIVERED_REASON property of messages in the queue has the value EXPIRED.

In the broker log file, look for any of the following messages: B2102, B2103, B2104. These messages all report that possible clock skew was detected.

To resolve the problem: Check that you are running a time synchronization program, as described in [“Preparing System Resources” on page 67](#).

Possible cause: An unexpected broker error has occurred.

To confirm this cause of the problem: Use the QBrowser demo application to inspect the contents of the dead message queue (see [“To Inspect the Dead Message Queue” on page 272](#)).

Check whether the JMS_SUN_DMQ_UNDELIVERED_REASON property of messages in the queue has the value ERROR.

To resolve the problem:

- Examine the broker log file to find the associated error.
- Contact Sun Technical Support to report the broker problem.

Possible cause: Consumers are not consuming messages before they time out.

To confirm this cause of the problem: Use the QBrowser demo application to inspect the contents of the dead message queue (see [“To Inspect the Dead Message Queue” on page 272](#)).

Check whether the JMS_SUN_DMQ_UNDELIVERED_REASON property of messages in the queue has the value EXPIRED.

Check to see if there are any consumers on the destination and the value for the Current Number of Active Consumers. For example:

```
imqcmd query dst -t q -n MyDest
```

If there are active consumers, then there might be any number of possible reasons why messages are timing out before being consumed. One is that the message timeout is too short for the speed at which the consumer executes. In that case, request that application developers increase message time-to-live values. Otherwise, investigate the following possible causes for messages to time out before being consumed:

Possible cause: There are too many producers for the number of consumers.

To confirm this cause of the problem: Use the QBrowser demo application to inspect the contents of the dead message queue (see [“To Inspect the Dead Message Queue” on page 272](#)).

Check whether the JMS_SUN_DMQ_UNDELIVERED_REASON property of messages in the queue has the value REMOVE_OLDEST or REMOVE_LOW_PRIORITY. If so, use the imqcmd query dst command to check the number of producers and consumers on the destination. If the number of producers exceeds the number of consumers, the production rate might be overwhelming the consumption rate.

To resolve the problem: Add more consumer clients or set the destination’s limit behavior to FLOW_CONTROL (which uses consumption rate to control production rate), using a command such as the following:

```
imqcmd update dst -n myDst -t q -o limitBehavior=FLOW_CONTROL
```

Possible cause: Producers are faster than consumers.

To confirm this cause of the problem: To determine whether slow consumers are causing producers to slow down, set the destination’s limit behavior to FLOW_CONTROL (which uses consumption rate to control production rate), using a command such as the following:

```
imqcmd update dst -n myDst -t q -o limitBehavior=FLOW_CONTROL
```

Use metrics to examine the destination’s input and output, using a command such as the following:

```
imqcmd metrics dst -n myDst -t q -m rts
```

In the metrics output, examine the following values:

- **Msgs/sec Out:** Shows how many messages per second the broker is removing. The broker removes messages when all consumers acknowledge receiving them, so the metric reflects consumption rate.
- **Msgs/sec In:** Shows how many messages per second the broker is receiving from producers. The metric reflects production rate.

Because flow control aligns production to consumption, note whether production slows or stops. If so, there is a discrepancy between the processing speeds of producers and consumers. You can also check the number of unacknowledged (UnAcked) messages sent, by using the `imqcmd list dst` command. If the number of unacknowledged messages is less than the size of the destination, the destination has additional capacity and is being held back by client flow control.

To resolve the problem: If production rate is consistently faster than consumption rate, consider using flow control regularly, to keep the system aligned. In addition, consider and attempt to resolve each of the following possible causes, which are subsequently described in more detail:

- [A consumer is too slow.](#)
- [Clients are not committing messages.](#)
- [Consumers are failing to acknowledge messages.](#)
- [Durable consumers are inactive.](#)
- [An unexpected broker error has occurred.](#)

Possible cause: A consumer is too slow.

To confirm this cause of the problem: Use `imqcmd metrics` to determine the rate of production and consumption, as described above under “[Producers are faster than consumers.](#)”

To resolve the problem:

- Set the destinations’ limit behavior to `FLOW_CONTROL`, using a command such as the following:

```
imqcmd update dst -n myDst -t q -o limitBehavior=FLOW_CONTROL
```

Use of flow control slows production to the rate of consumption and prevents the accumulation of messages in the destination. Producer applications hold messages until the destination can process them, with less risk of expiration.

- Find out from application developers whether producers send messages at a steady rate or in periodic bursts. If an application sends bursts of messages, increase destination limits as described in the next item.
- Increase destination limits based on number of messages or bytes, or both. To change the number of messages on a destination, enter a command with the following format:

```
imqcmd update dst -n destName -t {q|t} -o maxNumMsgs=number
```

To change the size of a destination, enter a command with the following format:

```
imqcmd update dst -n destName -t {q|t} -o maxTotalMsgBytes=number
```

Be aware that raising limits increases the amount of memory that the broker uses. If limits are too high, the broker could run out of memory and become unable to process messages.

- Consider whether you can accept loss of messages during periods of high production load.

Possible cause: Clients are not committing transactions.

To confirm this cause of the problem: Check with application developers to find out whether the application uses transactions. If so, list the active transactions as follows:

```
imqcmd list txn
```

Here is an example of the command output:

```
-----
Transaction ID      State    User name  # Msgs/# Acks  Creation time
-----
6800151593984248832  STARTED  guest      3/2           7/19/04 11:03:08 AM
```

Note the numbers of messages and number of acknowledgments. If the number of messages is high, producers may be sending individual messages but failing to commit transactions. Until the broker receives a commit, it cannot route and deliver the messages for that transaction. If the number of acknowledgments is high, consumers may be sending acknowledgments for individual messages but failing to commit transactions. Until the broker receives a commit, it cannot remove the acknowledgments for that transaction.

To resolve the problem: Contact application developers to fix the coding error.

Possible cause: Consumers are failing to acknowledge messages.

To confirm this cause of the problem: Contact application developers to determine whether the application uses system-based acknowledgment (AUTO_ACKNOWLEDGE or DUPES_ONLY) or client-based acknowledgment (CLIENT_ACKNOWLEDGE). If the application uses system-based acknowledgment, skip this section; if it uses client-based acknowledgment, first decrease the number of messages stored on the client, using a command like the following:

```
imqcmd update dst -n myDst -t q -o consumerFlowLimit=1
```

Next, you will determine whether the broker is buffering messages because a consumer is slow, or whether the consumer processes messages quickly but does not acknowledge them. List the destination, using the following command:

```
imqcmd list dst
```

After you supply a user name and password, output like the following appears:

Listing all the destinations on the broker specified by:

```
-----
Host          Primary Port
-----
localhost     7676
-----
Name          Type      State    Producers  Consumers  Msgs
```

				Total	Count	UnAck	Avg Size

MyDest	Queue	RUNNING	0	0	5	200	1177.0
mq.sys.dm	Queue	RUNNING	0	0	35	0	1422.0
Successfully listed destinations.							

The UnAck number represents messages that the broker has sent and for which it is waiting for acknowledgment. If this number is high or increasing, you know that the broker is sending messages, so it is not waiting for a slow consumer. You also know that the consumer is not acknowledging the messages.

To resolve the problem: Contact application developers to fix the coding error.

Possible cause: Durable subscribers are inactive.

To confirm this cause of the problem: Look at the topic’s durable subscribers, using the following command format:

`imqcmd list dur -d topicName`

To resolve the problem:

- Purge the durable subscribers using the `imqcmd purge dur` command.
- Restart the consumer applications.

▼ To Inspect the Dead Message Queue

A number of troubleshooting procedures involve an inspection of the dead message queue (`mq.sys.dm`). The following procedure explains how to carry out such an inspection by using the QBrowser demo application.

1 Locate the QBrowser demo application.

See [Appendix A, “Platform-Specific Locations of Message Queue Data,”](#) and look in the tables for “Example Applications and Locations.”

2 Run the QBrowser application.

Here is an example invocation on the Windows platform:

```
cd \MessageQueue3\demo\applications\qbrowser java QBrowser
```

The QBrowser main window appears.

3 Select the queue name `mq.sys.dm` and click Browse.

A list like the following appears:

QBrower 1.0 - localhost:7676

File

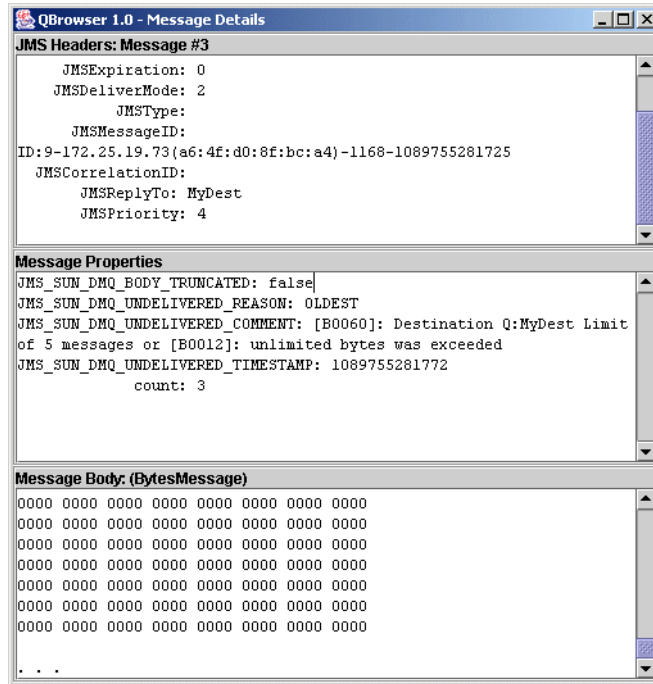
Queue Name: **mq.sys.dmq** Browse

#	Timestamp	Type	Mode	Priority
0	13/Jul/2004:14:48:01 PDT	BytesMessage	P	4
1	13/Jul/2004:14:48:01 PDT	BytesMessage	P	4
2	13/Jul/2004:14:48:01 PDT	BytesMessage	P	4
3	13/Jul/2004:14:48:01 PDT	BytesMessage	P	4
4	13/Jul/2004:14:48:01 PDT	BytesMessage	P	4
5	13/Jul/2004:14:48:01 PDT	BytesMessage	P	4
6	13/Jul/2004:14:48:01 PDT	BytesMessage	P	4
7	13/Jul/2004:14:48:01 PDT	BytesMessage	P	4
8	13/Jul/2004:14:48:01 PDT	BytesMessage	P	4
9	13/Jul/2004:14:48:01 PDT	BytesMessage	P	4
10	13/Jul/2004:14:48:01 PDT	BytesMessage	P	4
11	13/Jul/2004:14:48:01 PDT	BytesMessage	P	4
12	13/Jul/2004:14:48:01 PDT	BytesMessage	P	4
13	13/Jul/2004:14:48:01 PDT	BytesMessage	P	4
14	13/Jul/2004:14:48:01 PDT	BytesMessage	P	4
15	13/Jul/2004:14:48:01 PDT	BytesMessage	P	4
16	13/Jul/2004:14:48:01 PDT	BytesMessage	P	4
17	13/Jul/2004:14:48:01 PDT	BytesMessage	P	4
18	13/Jul/2004:14:48:01 PDT	BytesMessage	P	4
19	13/Jul/2004:14:48:01 PDT	BytesMessage	P	4
20	13/Jul/2004:14:53:50 PDT	BytesMessage	P	4
21	13/Jul/2004:14:53:50 PDT	BytesMessage	P	4
22	13/Jul/2004:14:53:50 PDT	BytesMessage	P	4
23	13/Jul/2004:14:53:50 PDT	BytesMessage	P	4
24	13/Jul/2004:14:53:50 PDT	BytesMessage	P	4
25	13/Jul/2004:14:53:50 PDT	BytesMessage	P	4
26	13/Jul/2004:14:53:50 PDT	BytesMessage	P	4
27	13/Jul/2004:14:53:50 PDT	BytesMessage	P	4
28	13/Jul/2004:14:53:50 PDT	BytesMessage	P	4

mq.sys.dmq: 35 Details...

4 Double-click any message to display details about that message:

The display should resemble the following:



You can inspect the Message Properties pane to determine the reason why the message was placed in the dead message queue.

PART III

Reference

- Chapter 15, “Command Line Reference”
- Chapter 16, “Broker Properties Reference”
- Chapter 17, “Physical Destination Property Reference”
- Chapter 18, “Administered Object Attribute Reference”
- Chapter 19, “JMS Resource Adapter Property Reference”
- Chapter 20, “Metrics Information Reference”
- Chapter 21, “JES Monitoring Framework Reference”

Command Line Reference

This chapter provides reference information on the use of the Message Queue command line administration utilities. It consists of the following sections:

- “Command Line Syntax” on page 277
- “Broker Utility” on page 278
- “Command Utility” on page 282
- “Object Manager Utility” on page 292
- “Database Manager Utility” on page 294
- “User Manager Utility” on page 295
- “Service Administrator Utility” on page 297
- “Key Tool Utility” on page 298

Command Line Syntax

Message Queue command line utilities are shell commands. The name of the utility is a command and its subcommands or options are arguments passed to that command. There is no need for separate commands to start or quit the utility.

All the command line utilities share the following command syntax:

utilityName [*subcommand*] [*commandArgument*] [[-*optionName* [*optionArgument*]] ...]

where *utilityName* is one of the following:

- `imqbrokerd` (Broker utility)
- `imqcmd` (Command utility)
- `imqobjmgr` (Object Manager utility)
- `imqdbmgr` (Database Manager utility)
- `imqusermgr` (User Manager utility)
- `imqsvcadm` (Service Administrator utility)
- `imqkeytool` (Key Tool utility)

Subcommands and command-level arguments, if any, must precede all options and their arguments; the options themselves may appear in any order. All subcommands, command arguments, options, and option arguments are separated with spaces. If the value of an option argument contains a space, the entire value must be enclosed in quotation marks. (It is generally safest to enclose any attribute-value pair in quotation marks.)

The following command, which starts the default broker, is an example of a command line with no subcommand clause:

```
imqbrokerd
```

Here is a fuller example:

```
imqcmd destroy dst -t q -n myQueue -u admin -f -s
```

This command destroys a queue destination (destination type q) named myQueue. Authentication is performed on the user name admin; the command will prompt for a password. The command will be performed without prompting for confirmation (-f option) and in silent mode, without displaying any output (-s option).

Broker Utility

The Broker utility (imqbrokerd) starts a broker. Command line options override values in the broker configuration files, but only for the current broker session.

Table 15–1 shows the options to the imqbrokerd command and the configuration properties, if any, overridden by each option.

TABLE 15–1 Broker Utility Options

Option	Properties Overridden	Description
-name <i>instanceName</i>	imq.instanceName	Instance name of broker Multiple broker instances running on the same host must have different instance names. Default value: imqbroker
-port <i>portNumber</i>	imq.portMapper.port	Port number for broker's Port Mapper Message Queue clients use this port number to connect to the broker. Multiple broker instances running on the same host must have different Port Mapper port numbers. Default value: 7676

TABLE 15-1 Broker Utility Options (Continued)

Option	Properties Overridden	Description
<code>-cluster broker1 [[,broker2] ...]</code>	<code>imq.cluster.brokerlist</code>	<p>Connect brokers into cluster¹</p> <p>The specified brokers are merged with the list in the <code>imq.cluster.brokerlist</code> property. Each broker argument has one of the forms</p> <p style="margin-left: 40px;"><i>hostName:portNumber</i></p> <p style="margin-left: 40px;"><i>hostName</i></p> <p style="margin-left: 40px;"><i>:portNumber</i></p> <p>If <i>hostName</i> is omitted, the default value is <code>localhost</code>; if <i>portNumber</i> is omitted, the default value is 7676.</p>
<code>-Dproperty=value</code>	Corresponding property in instance configuration file	<p>Set configuration property</p> <p>See Chapter 16, “Broker Properties Reference,” for information about broker configuration properties.</p> <p>Caution: Be careful to check the spelling and formatting of properties set with this option. Incorrect values will be ignored without notification or warning.</p>
<code>-reset props</code>	None	<p>Reset configuration properties</p> <p>Replaces the broker’s existing instance configuration file <code>config.properties</code> with an empty file; all properties assume their default values.</p>
<code>-reset store</code>	None	<p>Reset persistent data store</p> <p>Clears all persistent data from the data store (including persistent messages, durable subscriptions, and transaction information), allowing you to start the broker instance with a clean slate. To prevent the persistent store from being reset on subsequent restarts, restart the broker instance without the <code>-reset</code> option.</p> <p>To clear only persistent messages or durable subscriptions, use <code>-reset messages</code> or <code>-reset durables</code> instead.</p>
<code>-reset messages</code>	None	Clear persistent messages from data store
<code>-reset durables</code>	None	Clear durable subscriptions from data store

¹ Applies only to broker clusters

TABLE 15–1 Broker Utility Options (Continued)

Option	Properties Overridden	Description
- backup <i>fileName</i>	None	Back up configuration change record to file ¹ See “ Managing a Conventional Cluster's Configuration Change Record ” on page 182 for more information.
- restore <i>fileName</i>	None	Restore configuration change record from backup file ¹ The backup file must have been previously created using the - backup option. See “ Managing a Conventional Cluster's Configuration Change Record ” on page 182 for more information.
- remove instance	None	Remove broker instance ² Deletes the instance configuration file, log files, persistent store, and other files and directories associated with the instance.
- dbuser <i>userName</i>	imq.persist.jdbc.user	User name for JDBC-based persistent data store
- passfile <i>filePath</i>	imq.passfile.enabled imq.passfile.dirpath imq.passfile.name	Location of password file Sets the broker's imq.passfile.enabled property to true, imq.passfile.dirpath to the path containing the password file, and imq.passfile.name to the file name itself. See “ Password Files ” on page 168 for more information.
- shared	imq.jms.threadpool_model	Use shared thread pool model to implement jms connection service Execution threads will be shared among connections to increase the number of connections supported. Sets the broker's imq.jms.threadpool_model property to shared.
- javahome <i>path</i>	None	Location of alternative Java runtime Default behavior: Use runtime installed on system or bundled with Message Queue.

¹ Applies only to broker clusters² Requires user confirmation unless - force is also specified

TABLE 15-1 Broker Utility Options (Continued)

Option	Properties Overridden	Description
-vmargs <i>arg1</i> [[<i>arg2</i>] ...]	None	<p>Pass arguments to Java virtual machine</p> <p>Arguments are separated with spaces. To pass more than one argument, or an argument containing a space, enclose the argument list in quotation marks.</p> <p>VM arguments can be passed only from the command line; there is no associated configuration property in the instance configuration file.</p>
-startRmiRegistry	imq.jmx.rmiregistry.start	Start RMI registry at broker startup
-useRmiRegistry	imq.jmx.rmiregistry.use	Use external RMI registry
-rmiRegistryPort	imq.jmx.rmiregistry.port	Port number of RMI registry
-upgrade-store-nobackup	None	<p>Automatically remove old data store on upgrade to Message Queue 3.5 or 3.5 SPx from an incompatible version²</p> <p>See the <i>Message Queue Installation Guide</i> for more information.</p>
-force	None	<p>Perform action without user confirmation</p> <p>This option applies only to the -remove instance and -upgrade-store-nobackup options, which normally require confirmation.</p>
-loglevel <i>level</i>	imq.broker.log.level	<p>Logging level:</p> <p>NONE</p> <p>ERROR</p> <p>WARNING</p> <p>INFO</p> <p>Default value: INFO</p>
-metrics <i>interval</i>	imq.metrics.interval	Logging interval for broker metrics, in seconds
-tty	imq.log.console.output	<p>Log all messages to console</p> <p>Sets the broker's imq.log.console.output property to ALL.</p> <p>If not specified, only error and warning messages will be logged.</p>

² Requires user confirmation unless -force is also specified

TABLE 15-1 Broker Utility Options (Continued)

Option	Properties Overridden	Description
-s -silent	imq.log.console.output	Silent mode (no logging to console) Sets the broker's imq.log.console.output property to NONE.
-version	None	Display version information ³
-h -help	None	Display usage help ³

³ Any other options specified on the command line are ignored.

Command Utility

The Command utility (imqcmd) is used for managing brokers, connection services, connections, physical destinations, durable subscriptions, and transactions.

All imqcmd commands must include a subcommand (except those using the -v or -h option to display product version information or usage help, respectively). The possible subcommands are listed in [Table 15-2](#) and described in detail in the corresponding sections below. In addition, each imqcmd subcommand supports the general options shown in [“General Command Utility Options” on page 284](#).

Note – The -u *userName* option (and corresponding password) is required except when using the -v or -h option. Also if a subcommand accepts a broker address (-b option) and no host name or port number is specified, the values localhost and 7676 are assumed by default.

TABLE 15-2 Command Utility Subcommands

[“Broker Management” on page 285](#)

shutdown bkr	Shut down broker
restart bkr	Restart broker
pause bkr	Pause broker
quiesce bkr	Quiesce broker
unquiesce bkr	Unquiesce broker
resume bkr	Resume broker
takeover bkr	Initiate broker takeover
update bkr	Set broker properties

TABLE 15–2 Command Utility Subcommands (Continued)

reload cls	Reload cluster configuration
query bkr	List broker property values
list bkr	List brokers in cluster
metrics bkr	Display broker metrics
“Connection Service Management” on page 288	
pause svc	Pause connection service
resume svc	Resume connection service
update svc	Set connection service properties
list svc	List connection services available on broker
query svc	List connection service property values
metrics svc	Display connection service metrics
“Connection Management” on page 288	
list cxn	List connections on broker
query cxn	Display connection information
destroy cxn	Destroy connection
“Physical Destination Management” on page 289	
create dst	Create physical destination
destroy dst	Destroy physical destination
pause dst	Pause message delivery for physical destination
resume dst	Resume message delivery for physical destination
purge dst	Purge all messages from physical destination
compact dst	Compact physical destination
update dst	Set physical destination properties
list dst	List physical destinations
query dst	List physical destination property values
metrics dst	Display physical destination metrics
“Durable Subscription Management” on page 291	
destroy dur	Destroy durable subscription
purge dur	Purge all messages for durable subscription

TABLE 15-2 Command Utility Subcommands (Continued)

list dur	List durable subscriptions for topic
“Transaction Management” on page 291	
commit txn	Commit transaction
rollback txn	Roll back transaction
list txn	List transactions being tracked by broker
query txn	Display transaction information
list dur	List durable subscriptions for topic
“JMX Management” on page 292	
list jmx	List JMX service URLs of JMX connectors

General Command Utility Options

The additional options listed in [Table 15-3](#) can be applied to any subcommand of the `imqcmd` command.

TABLE 15-3 General Command Utility Options

Option	Description
-secure	Use secure connection to broker with <code>ssladmin</code> connection service
-u <i>userName</i>	User name for authentication If this option is omitted, the Command utility will prompt for it interactively.
-passfile <i>path</i>	Location of password file See “Password Files” on page 168 for more information.
-D	Set connection-related system property that affects how <code>imqcmd</code> creates a connection to the broker. Not used to set broker configuration properties. Usually overrides connection factory attributes for <code>imqcmd</code> client runtime. For example, the option in the following command changes the default value of <code>imqSSLIsTrusted</code> : <code>imqcmd list svc -secure -DimqSSLIsTrusted=true</code>

TABLE 15–3 General Command Utility Options (Continued)

Option	Description
- <i>rtm timeoutInterval</i>	Initial timeout interval, in seconds This is the initial length of time that the Command utility will wait for a reply from the broker before retrying a request. Each subsequent retry will use a timeout interval that is a multiple of this initial interval. Default value: 10.
- <i>rtr numRetries</i>	Number of retries to attempt after a broker request times out Default value: 5.
- <i>javahome path</i>	Location of alternative Java runtime Default behavior: Use runtime installed on system or bundled with Message Queue.
- <i>f</i>	Perform action without user confirmation
- <i>s</i>	Silent mode (no output displayed)
- <i>v</i>	Display version information ^{1,2}
- <i>h</i>	Display usage help ^{1,2}
- <i>H</i>	Display expanded usage help, including attribute list and examples ^{1,2}

¹ Any other options specified on the command line are ignored.² User name and password not needed

Broker Management

The Command utility cannot be used to start a broker; use the Broker utility (`imqbrokerd`) instead. Once the broker is started, you can use the `imqcmd` subcommands listed in [Table 15–4](#) to manage and control it.

TABLE 15-4 Command Utility Subcommands for Broker Management

Syntax	Description
<code>shutdown bkr [-b <i>hostName:portNumber</i>]</code> <code>[-time <i>nSeconds</i>]</code> <code>[-nofailover]</code>	<p>Shut down broker</p> <p>The <code>-time</code> option specifies the interval, in seconds, to wait before shutting down the broker. (The broker will not block, but will return immediately from the delayed shutdown request.) During the shutdown interval, the broker will not accept any new <code>jms</code> connections; <code>admin</code> connections will be accepted, and existing <code>jms</code> connections will continue to operate. A broker belonging to a high-availability cluster will not attempt to take over for any other broker during the shutdown interval.</p> <p>The <code>-nofailover</code> option indicates that no other broker is to take over the persistent data of the one being shut down. ¹</p>
<code>restart bkr [-b <i>hostName:portNumber</i>]</code>	<p>Restart broker</p> <p>Shuts down the broker and then restarts it using the same options specified when it was originally started.</p>
<code>pause bkr [-b <i>hostName:portNumber</i>]</code>	<p>Pause broker</p> <p>See “Pausing and Resuming a Broker” on page 89 for more information.</p>
<code>quiesce bkr [-b <i>hostName:portNumber</i>]</code>	<p>Quiesce broker</p> <p>The broker will stop accepting new connections; existing connections will continue to operate.</p>
<code>unquiesce bkr [-b <i>hostName:portNumber</i>]</code>	<p>Unquiesce broker</p> <p>The broker will resume accepting new connections, returning to normal operation.</p>
<code>resume bkr [-b <i>hostName:portNumber</i>]</code>	<p>Resume broker</p>

¹ Applies only to broker clusters

TABLE 15–4 Command Utility Subcommands for Broker Management (Continued)

Syntax	Description
takeover bkr -n <i>brokerID</i> [-f]	Initiate broker takeover ¹ Before taking over a broker, you should first shut it down manually using the shutdown bkr subcommand with the -nofailover option. If the specified broker appears to be still running, takeover bkr will display a confirmation message (Do you want to take over for this broker?). The -f option suppresses this message and initiates the takeover unconditionally. Note – The takeover bkr subcommand is intended only for use in failed-takeover situations. You should use it only as a last resort, and not as a general way of forcibly taking over a running broker.
update bkr [-b <i>hostName:portNumber</i>] -o <i>property1=value1</i> [[-o <i>property2=value2</i>] ...]	Set broker properties See Chapter 16, “Broker Properties Reference,” for information on broker properties.
reload cls	Reload cluster configuration ¹ Forces all persistent information to be brought up to date.
query bkr -b <i>hostName:portNumber</i>	List broker property values For brokers belonging to a cluster, also lists cluster properties such as broker list, master broker (for conventional clusters), and cluster identifier (for high-availability clusters).
list bkr	List brokers in cluster
metrics bkr [-b <i>hostName:portNumber</i>] [-m <i>metricType</i>] [-int <i>interval</i>] [-msp <i>numSamples</i>]	Display broker metrics The -m option specifies the type of metrics to display: ttl: Messages and packets flowing into and out of the broker rts: Rate of flow of messages and packets into and out of the broker per second cxn: Connections, virtual memory heap, and threads Default value: ttl. The -int option specifies the interval, in seconds, at which to display metrics. Default value: 5. The -msp option specifies the number of samples to display. Default value: Unlimited (infinite).

¹ Applies only to broker clusters

Connection Service Management

Table 15–5 lists the `imqcmd` subcommands for managing connection services.

TABLE 15–5 Command Utility Subcommands for Connection Service Management

Syntax	Description
<code>pause svc -n serviceName</code> <code>[-b hostName:portNumber]</code>	Pause connection service The <code>admin</code> connection service cannot be paused.
<code>resume svc -n serviceName</code> <code>[-b hostName:portNumber]</code>	Resume connection service
<code>update svc -n serviceName</code> <code>[-b hostName:portNumber]</code> <code>-o property1=value1</code> <code>[[-o property2=value2] ...]</code>	Set connection service properties See “ Connection Properties ” on page 299 for information on connection service properties.
<code>list svc [-b hostName:portNumber]</code>	List connection services available on broker
<code>query svc -n serviceName</code> <code>[-b hostName:portNumber]</code>	List connection service property values
<code>metrics svc -n serviceName</code> <code>[-b hostName:portNumber]</code> <code>[-m metricType]</code> <code>[-int interval]</code> <code>[-msp numSamples]</code>	Display connection service metrics The <code>-m</code> option specifies the type of metrics to display: <code>ttl</code> : Messages and packets flowing into and out of the broker by way of the specified connection service <code>rts</code> : Rate of flow of messages and packets into and out of the broker per second by way of the specified connection service <code>cxn</code> : Connections, virtual memory heap, and threads Default value: <code>ttl</code> . The <code>-int</code> option specifies the interval, in seconds, at which to display metrics. Default value: 5. The <code>-msp</code> option specifies the number of samples to display. Default value: Unlimited (infinite).

Connection Management

Table 15–6 lists the `imqcmd` subcommands for managing connections.

TABLE 15–6 Command Utility Subcommands for Connection Service Management

Syntax	Description
<code>list cxn [-svn <i>serviceName</i>] [-b <i>hostName:portNumber</i>]</code>	List connections on broker Lists all connections on the broker to the specified connection service. If no connection service is specified, all connections are listed.
<code>query cxn -n <i>connectionID</i> [-b <i>hostName:portNumber</i>]</code>	Display connection information
<code>destroy cxn -n <i>connectionID</i> [-b <i>hostName:portNumber</i>]</code>	Destroy connection

Physical Destination Management

Table 15–7 lists the `imqcmd` subcommands for managing physical destinations. In all cases, the `-t` (destination type) option can take either of two values:

q: Queue destination

t: Topic destination

TABLE 15–7 Command Utility Subcommands for Physical Destination Management

Syntax	Description
<code>create dst -t <i>destType</i> -n <i>destName</i> [[-o <i>property=value</i>] ...]</code>	Create physical destination ¹ The destination name <i>destName</i> may contain only alphanumeric characters (no spaces) and must begin with an alphabetic character or the underscore (<code>_</code>) or dollar sign (<code>\$</code>) character. It may not begin with the characters <code>mq</code> .
<code>destroy dst -t <i>destType</i> -n <i>destName</i></code>	Destroy physical destination ¹ This operation cannot be applied to a system-created destination, such as a dead message queue.

¹ Cannot be performed in a broker cluster whose master broker is temporarily unavailable

TABLE 15-7 Command Utility Subcommands for Physical Destination Management (Continued)

Syntax	Description
<pre>pause dst [-t <i>destType</i> -n <i>destName</i>] [-pst <i>pauseType</i>]</pre>	<p>Pause message delivery for physical destination</p> <p>Pauses message delivery for the physical destination specified by the -t and -n options. If these options are not specified, all destinations are paused.</p> <p>The -pst option specifies the type of message delivery to be paused:</p> <ul style="list-style-type: none"> PRODUCERS: Pause delivery from message producers CONSUMERS: Pause delivery to message consumers ALL: Pause all message delivery <p>Default value: ALL</p>
<pre>resume dst [-t <i>destType</i> -n <i>destName</i>]</pre>	<p>Resume message delivery for physical destination</p> <p>Resumes message delivery for the physical destination specified by the -t and -n options. If these options are not specified, all destinations are resumed.</p>
<pre>purge dst -t <i>destType</i> -n <i>destName</i></pre>	<p>Purge all messages from physical destination</p>
<pre>compact dst [-t <i>destType</i> -n <i>destName</i>]</pre>	<p>Compact physical destination</p> <p>Compacts the file-based persistent data store for the physical destination specified by the -t and -n options. If these options are not specified, all destinations are compacted.</p> <p>A destination must be paused before it can be compacted.</p>
<pre>update dst -t <i>destType</i> -n <i>destName</i> -o <i>property1=value1</i> [[-o <i>property2=value2</i>] ...]</pre>	<p>Set physical destination properties</p> <p>See Chapter 17, “Physical Destination Property Reference,” for information on physical destination properties.</p>
<pre>list dst [-t <i>destType</i>] [-tmp]</pre>	<p>List physical destinations</p> <p>Lists all physical destinations of the type specified by the -t option. If no destination type is specified, both queue and topic destinations are listed. If the -tmp option is specified, temporary destinations are listed as well.</p>
<pre>query dst -t <i>destType</i> -n <i>destName</i></pre>	<p>List physical destination property values</p>

TABLE 15–7 Command Utility Subcommands for Physical Destination Management (Continued)

Syntax	Description
<pre>metrics dst -t destType -n destName [-m metricType] [-int interval] [-msp numSamples]</pre>	<p>Display physical destination metrics</p> <p>The <code>-m</code> option specifies the type of metrics to display:</p> <ul style="list-style-type: none"> <code>ttl</code>: Messages and packets flowing into and out of the destination and residing in memory <code>rts</code>: Rate of flow of messages and packets into and out of the destination per second, along with other rate information <code>con</code>: Metrics related to message consumers <code>dsk</code>: Disk usage <p>Default value: <code>ttl</code>.</p> <p>The <code>-int</code> option specifies the interval, in seconds, at which to display metrics. Default value: 5.</p> <p>The <code>-msp</code> option specifies the number of samples to display. Default value: Unlimited (infinite).</p>

Durable Subscription Management

Table 15–8 lists the `imqcmd` subcommands for managing durable subscriptions.

TABLE 15–8 Command Utility Subcommands for Durable Subscription Management

Syntax	Description
<code>destroy dur -n subscriberName -c clientID</code>	Destroy durable subscription ¹
<code>purge dur -n subscriberName -c clientID</code>	Purge all messages for durable subscription
<code>list dur -d topicName</code>	List durable subscriptions for topic

¹ Cannot be performed in a broker cluster whose master broker is temporarily unavailable

Transaction Management

Table 15–9 lists the `imqcmd` subcommands for managing local (non-distributed) Message Queue transactions. Distributed transactions are managed by a distributed transaction manager rather than `imqcmd`.

TABLE 15–9 Command Utility Subcommands for Transaction Management

Syntax	Description
<code>commit txn -n transactionID</code>	Commit transaction
<code>rollback txn -n transactionID</code>	Roll back transaction
<code>list txn</code>	List transactions being tracked by broker
<code>query txn -n transactionID</code>	Display transaction information

JMX Management

The `imqcmd` subcommand shown in [Table 15–10](#) is used for administrative support of Java applications using the Java Management Extensions (JMX) application programming interface to configure and monitor Message Queue resources. See [Appendix D, “JMX Support,”](#) for further information on the broker's JMX support.

TABLE 15–10 Command Utility Subcommand for JMX Management

Syntax	Description
<code>list jmx</code>	List JMX service URLs of JMX connectors

Object Manager Utility

The Object Manager utility (`imqobjmgr`) creates and manages Message Queue administered objects. [Table 15–11](#) lists the available subcommands.

TABLE 15–11 Object Manager Subcommands

Subcommand	Description
<code>add</code>	Add administered object to object store
<code>delete</code>	Delete administered object from object store
<code>list</code>	List administered objects in object store
<code>query</code>	Display administered object information
<code>update</code>	Modify administered object

[Table 15–12](#) lists the options to the `imqobjmgr` command.

TABLE 15-12 Object Manager Options

Option	Description
-l <i>lookupName</i>	JNDI lookup name of administered object
-j <i>attribute=value</i>	Attributes of JNDI object store (see “ Object Stores ” on page 189)
-t <i>objectType</i>	Type of administered object: q: Queue destination t: Topic destination cf: Connection factory qf: Queue connection factory tf: Topic connection factory xcf: Connection factory for distributed transactions xqf: Queue connection factory for distributed transactions xtf: Topic connection factory for distributed transactions
-o <i>attribute=value</i>	Attributes of administered object (see “ Administered Object Attributes ” on page 192 and Chapter 18 , “ Administered Object Attribute Reference ”)
-r <i>readOnlyState</i>	Is administered object read-only? If true, client cannot modify object’s attributes. Default value: false.
-i <i>fileName</i>	Name of command file containing all or part of subcommand clause
-pre	Preview results without performing command This option is useful for checking the values of default attributes.
-javahome <i>path</i>	Location of alternative Java runtime Default behavior: Use runtime installed on system or bundled with Message Queue.
-f	Perform action without user confirmation
-s	Silent mode (no output displayed)
-v	Display version information ¹
-h	Display usage help ¹
-H	Display expanded usage help, including attribute list and examples ¹

¹ Any other options specified on the command line are ignored.

Database Manager Utility

The Database Manager utility (`imqdbmgr`) sets up the database schema for a JDBC-based data store. You can also use it to delete Message Queue database tables that have become corrupted, change the database, display information about the database, convert a standalone database for use in a high-availability broker cluster, or back up and restore a highly-available database.

Table 15–13 lists the `imqdbmgr` subcommands.

TABLE 15–13 Database Manager Subcommands

Subcommand	Description
<code>create all</code>	Create new database and persistent data store schema Used on embedded database systems. The broker property <code>imq.persist.jdbc.vendorName.createdburl</code> must be specified.
<code>create tbl</code>	Create persistent data store schema for existing database Used on external database systems. For brokers belonging to a high-availability broker cluster (<code>imq.cluster.ha = true</code>), the schema created is for the cluster's shared data store, in accordance with the database vendor identified by the broker's <code>imq.persist.jdbc.dbVendor</code> property. If <code>imq.cluster.ha = false</code> , the schema is for the individual broker's standalone data store. Since the two types of data store can coexist in the same database, they are distinguished by appending a suffix to all table names: <div><i>C clusterID</i>: Shared data store <i>S brokerID</i>: Standalone data store</div>
<code>delete tbl</code>	Delete Message Queue database tables from current data store
<code>delete oldtbl</code>	Delete Message Queue database tables from earlier-version data store Used after the data store has been automatically migrated to the current version of Message Queue.
<code>recreate tbl</code>	Re-create persistent store schema Deletes all existing Message Queue database tables from the current persistent store and then re-creates the schema.
<code>query</code>	Display information about the data store
<code>upgrade hystore</code>	Upgrade standalone data store to high-availability shared data store
<code>backup</code>	Back up JDBC-based data store to backup files
<code>restore</code>	Restore JDBC-based data store from backup files

TABLE 15–13 Database Manager Subcommands (Continued)

Subcommand	Description
remove bkr	Remove broker from shared data store The broker must not be running.
reset lck	Reset data store lock Resets the lock so that the database can be used by other processes.

Table 15–14 lists the options to the `imqdbmgr` command.

TABLE 15–14 Database Manager Options

Option	Description
-b <i>instanceName</i>	Instance name of broker
-D <i>property=value</i>	Set broker configuration property See “ Persistence Properties ” on page 307 for information about persistence-related broker configuration properties. Caution: Be careful to check the spelling and formatting of properties set with this option. Incorrect values will be ignored without notification or warning.
-u <i>userName</i>	User name for authentication against the database
-passfile <i>filePath</i>	Location of password file See “ Password Files ” on page 168 for more information.
-n <i>brokerID</i>	Broker identifier of broker to be removed from shared data store
-dir <i>dirPath</i>	Backup directory for backing up or restoring JDBC-based data store
-v	Display version information ¹
-h	Display usage help ¹

¹ Any other options specified on the command line are ignored.

User Manager Utility

The User Manager utility (`imusermgr`) is used for populating or editing a flat-file user repository. The utility must be run on the same host where the broker is installed; if a broker-specific user repository does not yet exist, you must first start up the corresponding broker instance in order to create it. You will also need the appropriate permissions to write to the repository: on the Solaris or Linux platforms, this means you must be either the root user or the user who originally created the broker instance.

Table 15–15 lists the subcommands available with the `imqusermgr` command. In all cases, the `-i` option specifies the instance name of the broker to whose user repository the command applies; if not specified, the default name `imqbroker` is assumed.

TABLE 15–15 User Manager Subcommands

Syntax	Description
<code>add [-i <i>instanceName</i>]</code> <code>-u <i>userName</i> -p <i>password</i></code> <code>[-g <i>group</i>]</code>	Add user and password to repository The optional <code>-g</code> option specifies a group to which to assign this user: <div>admin user anonymous</div>
<code>delete [-i <i>instanceName</i>]</code> <code>-u <i>userName</i></code>	Delete user from repository
<code>update [-i <i>instanceName</i>]</code> <code>-u <i>userName</i> -p <i>password</i></code> <code>update [-i <i>instanceName</i>]</code> <code>-u <i>userName</i> -a <i>activeStatus</i></code> <code>update [-i <i>instanceName</i>]</code> <code>-u <i>userName</i> -p <i>password</i></code> <code>-a <i>activeStatus</i></code>	Set user's password or active status (or both) The <code>-a</code> option takes a boolean value specifying whether to make the user active (<code>true</code>) or inactive (<code>false</code>). An inactive status means that the user entry remains in the user repository, but the user will not be authenticated, even if using the correct password. Default value: <code>true</code> .
<code>list [-i <i>instanceName</i>]</code> <code>[-u <i>userName</i>]</code>	Display user information If no user name is specified, all users in the repository are listed.

In addition, the options listed in Table 15–16 can be applied to any subcommand of the `imqusermgr` command.

TABLE 15–16 General User Manager Options

Option	Description
<code>-f</code>	Perform action without user confirmation
<code>-s</code>	Silent mode (no output displayed)
<code>-v</code>	Display version information ¹
<code>-h</code>	Display usage help ¹

¹ Any other options specified on the command line are ignored.

Service Administrator Utility

The Service Administrator utility (`imqsvcadmin`) installs a broker as a Windows service. [Table 15–17](#) lists the available subcommands.

TABLE 15–17 Service Administrator Subcommands

Subcommand	Description
<code>install</code>	Install service
<code>remove</code>	Remove service
<code>query</code>	Display startup options Startup options can include whether the service is started manually or automatically, its location, the location of the Java runtime, and the values of arguments passed to the broker on startup (see Table 15–18).

[Table 15–18](#) lists the options to the `imqsvcadmin` command.

TABLE 15–18 Service Administrator Options

Option	Description
<code>-javahome <i>path</i></code>	Location of alternative Java runtime Default behavior: Use runtime installed on system or bundled with Message Queue.
<code>-jrehome <i>path</i></code>	Location of alternative Java Runtime Environment (JRE)
<code>-vmargs <i>arg1</i> [<i>arg2</i>] ...]</code>	Additional arguments to pass to Java Virtual Machine (JVM) running broker service ¹ Example: <code>imqsvcadmin install -vmargs "-Xms16m -Xmx128m"</code>
<code>-args <i>arg1</i> [<i>arg2</i>] ...]</code>	Additional command line arguments to pass to broker service ¹ Example: <code>imqsvcadmin install -args "-passfile d:\\imqpassfile"</code> See “ Broker Utility ” on page 278 for information about broker command line arguments.
<code>-h</code>	Display usage help ²

¹ These arguments can also be specified in the Start Parameters field under the General tab in the service’s Properties window (reached by way of the Services tool in the Windows Administrative Tools control panel).

² Any other options specified on the command line are ignored.

Any information you specify using the `-javahome`, `-vmargs`, and `-args` options is stored in the Windows registry under the keys `JREHome`, `JVMArgs`, and `ServiceArgs` in the path

`HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\imQ_Broker\Parameters`

Key Tool Utility

The Key Tool utility (`imqkeytool`) generates a self-signed certificate for the broker, which can be used for the `ssljms`, `ssladmin`, or `cluster` connection service. The syntax is

```
imqkeytool -broker
```

On UNIX systems, you might need to run the utility from the root user account.

Broker Properties Reference

This chapter provides reference information about configuration properties for a message broker. It consists of the following sections:

- “Connection Properties” on page 299
- “Routing and Delivery Properties” on page 302
- “Persistence Properties” on page 307
- “Security Properties” on page 310
- “Monitoring Properties” on page 316
- “Cluster Configuration Properties” on page 321
- “JMX Properties” on page 324
- “Alphabetical List of Broker Properties” on page 326

Connection Properties

Table 16–1 lists the broker properties related to connection services.

TABLE 16–1 Broker Connection Properties

Property	Type	Default Value	Description
<code>imq.brokerid</code>	String	None	<p>Broker identifier</p> <p>For brokers using a shared JDBC-based data store, this string is appended to the names of all database tables to identify each table with a particular broker.</p> <p>Must be a unique alphanumeric string of no more than $n - 13$ characters, where n is the maximum table name length allowed by the database.</p> <p>This property is unnecessary for an embedded database or a standalone database which stores data for only one broker instance.</p> <p>Note – For high-availability broker clusters (<code>imq.cluster.ha = true</code>), database table names also use the <code>imq.cluster.clusterid</code> property (see Table 16–11).</p>
<code>imq.service.activelist</code> ¹	String	<code>jms, admin</code>	<p>List of connection services to be activated at broker startup, separated by commas</p> <p>See Table 6–1 under “Configuring Connection Services” on page 93 for the names of the available connection services.</p>
<code>imq.hostname</code>	String	All available IP addresses	Default host name or IP address for all connection services
<code>imq.portmapper.hostname</code>	String	None	<p>Host name or IP address of Port Mapper</p> <p>If specified, overrides <code>imq.hostname</code>. This might be necessary, for instance, if the broker’s host computer has more than one network interface card installed.</p>
<code>imq.portmapper.port</code> ²	Integer	7676	<p>Port number of Port Mapper</p> <p>Note – If multiple broker instances are running on the same host, each must be assigned a unique Port Mapper port.</p>
<code>imq.serviceName.protocolType.hostname</code> ³	String	None	<p>Host name or IP address for connection service</p> <p>If specified, overrides <code>imq.hostname</code> for the designated connection service. This might be necessary, for instance, if the broker’s host computer has more than one network interface card installed.</p>

¹ Must have the same value for all brokers in a high-availability cluster.² Can be used with `imqcmd update bkr` command³ `jms`, `ssljms`, `admin`, and `ssladmin` services only; see [Appendix C, “HTTP/HTTPS Support,”](#) for information on configuring the `httpjms` and `httpsjms` services

TABLE 16–1 Broker Connection Properties (Continued)

Property	Type	Default Value	Description
<code>imq.serviceName.protocolType.port</code> ³	Integer	0	Port number for connection service A value of 0 specifies that the port number should be allocated dynamically by the Port Mapper. You might need to set a different value, for instance, to specify a static port number for connecting to the broker through a firewall.
<code>imq.portmapper.backlog</code>	Integer	50	Maximum number of pending Port Mapper requests in operating system backlog
<code>imq.serviceName.threadpool_model</code> ⁴	String	dedicated	Threading model for thread pool management: dedicated: Two dedicated threads per connection, one for incoming and one for outgoing messages shared: Connections processed by shared thread when sending or receiving messages The dedicated model limits the number of connections that can be supported, but provides higher performance; the shared model increases the number of possible connections, but at the cost of lower performance because of the additional overhead needed for thread management.
<code>imq.serviceName.min_threads</code>	Integer	jms: 10 ssljms: 10 httpjms: 10 httpsjms: 10 admin: 4 ssladmin: 4	Minimum number of threads maintained in connection service's thread pool When the number of available threads exceeds this threshold, threads will be shut down as they become free until the minimum is reached. The default value varies by connection service, as shown.
<code>imq.serviceName.max_threads</code>	Integer	jms: 1000 ssljms: 500 httpjms: 500 httpsjms: 500 admin: 10 ssladmin: 10	Number of threads beyond which no new threads are added to the thread pool for use by the named connection service Must be greater than 0 and greater than the value of <code>imq.serviceName.min_threads</code> . The default value varies by connection service, as shown.

³ jms, ssljms, admin, and ssladmin services only; see [Appendix C, “HTTP/HTTPS Support,”](#) for information on configuring the httpjms and httpsjms services⁴ jms and admin services only

TABLE 16–1 Broker Connection Properties (Continued)

Property	Type	Default Value	Description
imq.shared.connectionMonitor_limit ⁵	Integer	Solaris: 512 Linux: 512 Windows: 64	Maximum number of connections monitored by a distributor thread The system allocates enough distributor threads to monitor all connections. The smaller the value of this property, the faster threads can be assigned to active connections. A value of –1 denotes an unlimited number of connections per thread. The default value varies by operating-system platform, as shown.
imq.ping.interval	Integer	120	Interval, in seconds, at which to test connection between client and broker A value of 0 or –1 disables periodic testing of the connection.

⁵ Shared threading model only

Routing and Delivery Properties

Table 16–2 lists the broker properties related to routing and delivery services. Properties that configure the automatic creation of destinations are listed in Table 16–3.

TABLE 16–2 Broker Routing and Delivery Properties

Property	Type	Default Value	Description
imq.system.max_count ¹	Integer	–1	Maximum number of messages held by broker A value of –1 denotes an unlimited message count.
imq.system.max_size ¹	String	–1	Maximum total size of messages held by broker The value may be expressed in bytes, kilobytes, or megabytes, using the following suffixes: b: Bytes k: Kilobytes (1024 bytes) m: Megabytes (1024 × 1024 = 1,048,576 bytes) An unsuffixed value is expressed in bytes; a value of –1 denotes an unlimited message capacity.

¹ Can be used with imqcmd update bkr command

TABLE 16–2 Broker Routing and Delivery Properties (Continued)

Property	Type	Default Value	Description
			Examples: 1600: 1600 bytes 1600b: 1600 bytes 16k: 16 kilobytes (= 16,384 bytes) 16m: 16 megabytes (= 16,777,216 bytes) –1: No limit
<code>imq.message.max_size¹</code>	String	70m	Maximum size of a single message body The syntax is the same as for <code>imq.system.max_size</code> (see above).
<code>imq.message.expiration.interval</code>	Integer	60	Interval, in seconds, at which expired messages are reclaimed
<code>imq.resourceState.threshold</code>	Integer	green: 0 yellow: 80 orange: 90 red: 98	Percent utilization at which memory resource state is triggered (where <i>resourceState</i> is green, yellow, orange, or red)
<code>imq.resourceState.count</code>	Integer	green: 5000 yellow: 500 orange: 50 red: 0	Maximum number of incoming messages allowed in a batch before checking whether memory resource state threshold has been reached (where <i>resourceState</i> is green, yellow, orange, or red) This limit throttles back message producers as system memory becomes increasingly scarce.
<code>imq.destination.DMQ.truncateBody¹</code>	Boolean	false	Remove message body before storing in dead message queue? If true, only the message header and property data will be saved.
<code>imq.transaction.autorollback</code>	Boolean	false	Automatically roll back distributed transactions left in prepared state at broker startup? If false, transactions must be manually committed or rolled back using the Command utility (<code>imqcmd</code>).

¹ Can be used with `imqcmd update bkr` command

TABLE 16–3 Broker Properties for Auto-Created Destinations

Property	Type	Default Value	Description
<code>imq.autocreate.queue^{1,2}</code>	Boolean	true	Allow auto-creation of queue destinations?
<code>imq.autocreate.topic³</code>	Boolean	true	Allow auto-creation of topic destinations?

¹ Can be used with `imqcmd update bkr` command

² Queue destinations only

³ Topic destinations only

TABLE 16-3 Broker Properties for Auto-Created Destinations (Continued)

Property	Type	Default Value	Description
<code>imq.autocreate.destination.maxNumMsgs</code>	Integer	<code>100000</code>	<p>Maximum number of unconsumed messages</p> <p>A value of <code>-1</code> denotes an unlimited number of messages.</p> <p>Note – When flow control is in effect (<code>imq.autocreate.destination.limitBehavior = FLOW_CONTROL</code>), it is possible for the specified message limit to be exceeded because the broker cannot react quickly enough to stop the flow of incoming messages. In such cases, the value specified for <code>imq.autocreate.destination.maxNumMsgs</code> serves as merely a hint for the broker rather than a strictly enforced limit.</p>
<code>imq.autocreate.destination.maxBytesPerMsg</code>	String	<code>10k</code>	<p>Maximum size, in bytes, of any single message</p> <p>The value may be expressed in bytes, kilobytes, or megabytes, using the following suffixes:</p> <ul style="list-style-type: none"><code>b</code>: Bytes<code>k</code>: Kilobytes (1024 bytes)<code>m</code>: Megabytes (1024 × 1024 = 1,048,576 bytes) <p>An unsuffixed value is expressed in bytes; a value of <code>-1</code> denotes an unlimited message size.</p> <p>Examples:</p> <ul style="list-style-type: none"><code>1600</code>: 1600 bytes<code>1600b</code>: 1600 bytes<code>16k</code>: 16 kilobytes (= 16,384 bytes)<code>16m</code>: 16 megabytes (= 16,777,216 bytes)<code>-1</code>: No limit
<code>imq.autocreate.destination.maxTotalMsgBytes</code>	String	<code>10m</code>	<p>Maximum total memory, in bytes, for unconsumed messages</p> <p>The syntax is the same as for <code>imq.autocreate.destination.maxBytesPerMsg</code> (see above).</p>

TABLE 16-3 Broker Properties for Auto-Created Destinations (Continued)

Property	Type	Default Value	Description
<code>imq.autocreate.destination.limitBehavior</code>	String	REJECT_NEWEST	<p>Broker behavior when memory-limit threshold reached:</p> <p>FLOW_CONTROL: Slow down producers</p> <p>REMOVE_OLDEST: Throw out oldest messages</p> <p>REMOVE_LOW_PRIORITY: Throw out lowest-priority messages according to age; no notification to producing client</p> <p>REJECT_NEWEST: Reject newest messages; notify producing client with an exception only if message is persistent</p> <p>If the value is REMOVE_OLDEST or REMOVE_LOW_PRIORITY and the <code>imq.autocreate.destination.useDMQ</code> property is true, excess messages are moved to the dead message queue.</p>
<code>imq.autocreate.destination.maxNumProducers</code>	Integer	100	<p>Maximum number of message producers for destination</p> <p>When this limit is reached, no new producers can be created. A value of -1 denotes an unlimited number of producers.</p>
<code>imq.autocreate.queue.maxNumActiveConsumers</code> ²	Integer	-1	<p>Maximum number of active message consumers in load-balanced delivery from queue destination</p> <p>A value of -1 denotes an unlimited number of consumers.</p>
<code>imq.autocreate.queue.maxNumBackupConsumers</code> ²	Integer	0	<p>Maximum number of backup message consumers in load-balanced delivery from queue destination</p> <p>A value of -1 denotes an unlimited number of consumers.</p>

² Queue destinations only

TABLE 16–3 Broker Properties for Auto-Created Destinations (Continued)

Property	Type	Default Value	Description
<code>imq.autocreate.queue.consumerFlowLimit</code> ²	Integer	1000	<p>Maximum number of messages delivered to queue consumer in a single batch</p> <p>In load-balanced queue delivery, this is the initial number of queued messages routed to active consumers before load balancing begins. A destination consumer can override this limit by specifying a lower value on a connection.</p> <p>A value of –1 denotes an unlimited number of messages.</p>
<code>imq.autocreate.topic.consumerFlowLimit</code> ³	Integer	1000	<p>Maximum number of messages delivered to topic consumer in a single batch</p> <p>A value of –1 denotes an unlimited number of consumers.</p>
<code>imq.autocreate.destination.isLocalOnly</code>	Boolean	false	<p>Local delivery only?</p> <p>This property applies only to destinations in broker clusters, and cannot be changed once the destination has been created. If <code>true</code>, the destination is not replicated on other brokers and is limited to delivering messages only to local consumers (those connected to the broker on which the destination is created).</p>
<code>imq.autocreate.queue.localDeliveryPreferred</code> ²	Boolean	false	<p>Local delivery preferred?</p> <p>This property applies only to load-balanced queue delivery in broker clusters. If <code>true</code>, messages will be delivered to remote consumers only if there are no consumers on the local broker; the destination must not be restricted to local-only delivery (<code>imq.autocreate.destination.isLocalOnly</code> must be <code>false</code>).</p>
<code>imq.autocreate.destination.useDMQ</code>	Boolean	true	<p>Send dead messages to dead message queue?</p> <p>If <code>false</code>, dead messages will simply be discarded.</p>
<code>validateXMLSchemaEnabled</code>	Boolean	false	<p>XML schema validation is enabled?</p> <p>If set to <code>false</code> or not set, then XML schema validation is not enabled for the destination.</p>

² Queue destinations only³ Topic destinations only

TABLE 16–3 Broker Properties for Auto-Created Destinations (Continued)

Property	Type	Default Value	Description
XMLSchemaURLList	String	null	<p>Space separated list of XML schema document (XSD) URI strings</p> <p>The URIs point to the location of one or more XSDs to use for XML schema validation, if enabled.</p> <p>Use double quotes around this value if multiple URIs are specified.</p> <p>Example:</p> <p>“http://foo/flap.xsd http://test.com/test.xsd”</p> <p>If this property is not set or null and XML validation is enabled, XML validation is performed using a DTD specified in the XML document.</p>
reloadXMLSchemaOnFailure	Boolean	false	<p>Reload XML schema on failure enabled?</p> <p>If set to false or not set, then the schema is not reloaded if validation fails.</p>

Persistence Properties

Message Queue supports both file-based and JDBC-based persistence modules. The broker property `imq.persist.store` (Table 16–4) specifies which module to use. The following sections describe the broker configuration properties for the two modules.

TABLE 16–4 Global Broker Persistence Property

Property	Type	Default Value	Description
<code>imq.persist.store</code>	String	file	<p>Module used for persistent data storage:</p> <ul style="list-style-type: none"> <code>file</code>: File-based persistence <code>jdbc</code>: JDBC-based persistence <p>Must be set to <code>jdbc</code> for high-availability broker clusters (<code>imq.cluster.ha = true</code>).</p>

File-Based Persistence Properties

Table 16–5 lists the broker properties related to file-based persistence.

TABLE 16-5 Broker Properties for File-Based Persistence

Property	Type	Default Value	Description
imq.persist.file.message.max_record_size	String	1m	<p>Maximum-size message to add to message storage file</p> <p>Any message exceeding this size will be stored in a separate file of its own.</p> <p>The value may be expressed in bytes, kilobytes, or megabytes, using the following suffixes:</p> <ul style="list-style-type: none">b: Bytesk: Kilobytes (1024 bytes)m: Megabytes (1024 × 1024 = 1,048,576 bytes) <p>An unsuffixed value is expressed in bytes.</p> <p>Examples:</p> <ul style="list-style-type: none">1600: 1600 bytes1600b: 1600 bytes16k: 16 kilobytes (= 16,384 bytes)16m: 16 megabytes (= 16,777,216 bytes)
imq.persist.file.destination.message.filepool.limit	Integer	100	<p>Maximum number of free files available for reuse in destination file pool</p> <p>Free files in excess of this limit will be deleted. The broker will create and delete additional files in excess of the limit as needed.</p> <p>The higher the limit, the faster the broker can process persistent data.</p>
imq.persist.file.message.filepool.cleanratio	Integer	0	<p>Percentage of files in free file pools to be maintained in a clean (empty) state</p> <p>The higher this value, the less disk space is required for the file pool, but the more overhead is needed to clean files during operation.</p>
imq.persist.file.message.cleanup	Boolean	false	<p>Clean up files in free file pools on shutdown?</p> <p>Setting this property to true saves disk space for the file store, but slows broker shutdown.</p>

TABLE 16–5 Broker Properties for File-Based Persistence (Continued)

Property	Type	Default Value	Description
<code>imq.persist.file.sync.enabled</code>	Boolean	false	<p>Synchronize in-memory state with physical storage device?</p> <p>Setting this property to <code>true</code> eliminates data loss due to system crashes, but at a cost in performance.</p> <p>Note – If running Sun Cluster and the Sun Cluster Data Service for Message Queue, set this property to <code>true</code> for brokers on all cluster nodes.</p>
<code>imq.persist.file.transaction.memorymappedfile.enabled</code>	Boolean	true	<p>Use memory-mapped file to store transaction data?</p> <p>Setting this property to <code>true</code> improves performance at the cost of increased memory usage. Set to <code>false</code> for file systems that do not support memory-mapped files.</p>

JDBC-Based Persistence Properties

Table 16–6 lists the broker properties related to JDBC-based persistence. The first of these properties, `imq.persist.jdbc.dbVendor`, identifies the database vendor being used for the broker's persistent data store; all of the remaining properties are qualified by this vendor name.

TABLE 16–6 Broker Properties for JDBC-Based Persistence

Property	Type	Default Value	Description
<code>imq.persist.jdbc.dbVendor</code>	String	None	<p>Name of database vendor for persistent data store:</p> <p><code>hadb</code>: HADB (Sun Microsystems, Inc.)</p> <p><code>derby</code>: Java DB (Sun Microsystems, Inc.)</p> <p><code>oracle</code>: Oracle (Oracle Corporation)</p> <p><code>mysql</code>: MySQL (Sun Microsystems, Inc.)</p> <p><code>postgresql</code>: PostgreSQL</p>
<code>imq.persist.jdbc.vendorName.driver</code>	String	per vendor	Java class name of JDBC driver, if needed, for connecting to database from vendor <i>vendorName</i>
<code>imq.persist.jdbc.vendorName.opendburl</code>	String	None	<p>URL for connecting to existing database from vendor <i>vendorName</i></p> <p>Applicable when driver is used to connect to database.</p>

TABLE 16–6 Broker Properties for JDBC-Based Persistence (Continued)

Property	Type	Default Value	Description
<code>imq.persist.jdbc.vendorName.createdburl</code> ¹	String	None	URL for creating new database from vendor <i>vendorName</i> Applies for embedded database, such as Java DB.
<code>imq.persist.jdbc.vendorName.closedburl</code> ¹	String	None	URL for closing connection to database from vendor <i>vendorName</i> Applies for some embedded databases, such as Java DB.
<code>imq.persist.jdbc.vendorName.user</code> ¹	String	None	User name, if required, for connecting to database from vendor <i>vendorName</i> For security reasons, the value can instead be specified using command line options <code>imqbrokerd -dbuser</code> and <code>imqdbmgr -u</code> .
<code>imq.persist.jdbc.vendorName.needpassword</code> ¹	Boolean	false	Does database from vendor <i>vendorName</i> require a password for broker access? If true, the <code>imqbrokerd</code> and <code>imqdbmgr</code> commands will prompt for a password, unless you use the <code>-passfile</code> option to specify a password file containing it.
<code>imq.persist.jdbc.vendorName.password</code> ^{1,2}	String	None	Password, if required, for connecting to database from vendor <i>vendorName</i>
<code>imq.persist.jdbc.vendorName.property.propName</code> ¹	String	None	Vendor-specific property <i>propName</i> for database from vendor <i>vendorName</i>
<code>imq.persist.jdbc.vendorName.tableoption</code> ¹	String	None	Vendor-specific options passed to the database when creating the table schema.

¹ Optional

² Should be used only in password files

Security Properties

Table 16–7 lists broker properties related to security services: authentication, authorization, and encryption. Table 16–8 lists broker properties related specifically to LDAP-based authentication, and Table 16–9 lists broker properties related specifically to JAAS-based authentication.

TABLE 16-7 Broker Security Properties

Property	Type	Default Value	Description
<code>imq.authentication.basic.user_repository</code>	String	<code>file</code>	Type of user authentication: <code>file</code> : File-based <code>ldap</code> : Lightweight Directory Access Protocol <code>jaas</code> : Java Authentication and Authorization Service
<code>imq.authentication.type</code>	String	<code>digest</code>	Password encoding method: <code>digest</code> : MD5 (for file-based authentication) <code>basic</code> : Base-64 (for LDAP or JAAS authentication)
<code>imq.serviceName.authentication.type</code>	String	<code>None</code>	Password encoding method for connection service <code>serviceName</code> : <code>digest</code> : MD5 (for file-based authentication) <code>basic</code> : Base-64 (for LDAP or JAAS authentication) If specified, overrides <code>imq.authentication.type</code> for the designated connection service.
<code>imq.authentication.client.response.timeout</code>	Integer	<code>180</code>	Interval, in seconds, to wait for client response to authentication requests
<code>imq.accesscontrol.enabled</code>	Boolean	<code>true</code>	Use access control? If <code>true</code> , the system will check the access control file to verify that an authenticated user is authorized to use a connection service or to perform specific operations with respect to specific destinations.
<code>imq.accesscontrol.type</code>	String	<code>file</code>	Specifies the access control type

TABLE 16–7 Broker Security Properties (Continued)

Property	Type	Default Value	Description
<code>imq.serviceName.accesscontrol.enabled</code>	Boolean	None	<p>Use access control for connection service?</p> <p>If specified, overrides <code>imq.accesscontrol.enabled</code> for the designated connection service.</p> <p>If <code>true</code>, the system will check the access control file to verify that an authenticated user is authorized to use the designated connection service or to perform specific operations with respect to specific destinations.</p>
<code>imq.accesscontrol.file.filename</code>	String	<code>accesscontrol.properties</code>	<p>Name of access control file</p> <p>The file name specifies a path relative to the access control directory (see Appendix A, “Platform-Specific Locations of Message Queue Data”).</p>
<code>imq.serviceName.accesscontrol.file.filename</code>	String	None	<p>Name of access control file for connection service</p> <p>If specified, overrides <code>imq.accesscontrol.file.filename</code> for the designated connection service.</p> <p>The file name specifies a path relative to the access control directory (see Appendix A, “Platform-Specific Locations of Message Queue Data”).</p>
<code>imq.accesscontrol.file.url</code>	String	Not set	<p>The location, as a URL, of the access control file.</p>
<code>imq.serviceName.accesscontrol.file.url</code>	String	None	<p>The location, as a URL, of the access control file for connection service</p> <p>If specified, overrides <code>imq.accesscontrol.file.url</code> for the designated connection service.</p>

TABLE 16–7 Broker Security Properties (Continued)

Property	Type	Default Value	Description
imq.keystore.file.dirpath	String	See Appendix A, “Platform-Specific Locations of Message Queue Data”	Path to directory containing key store file
imq.keystore.file.name	String	keystore	Name of key store file
imq.keystore.password ¹	String	None	Password for key store file
imq.passfile.enabled	Boolean	false	Obtain passwords from password file?
imq.passfile.dirpath	String	See Appendix A, “Platform-Specific Locations of Message Queue Data”	Path to directory containing password file
imq.passfile.name	String	passfile	Name of password file
imq.imqcmd.password ¹	String	None	Password for administrative user The Command utility (imqcmd) uses this password to authenticate the user before executing a command.

¹ To be used only in password files

[Table 16–8](#) lists broker properties related to LDAP-based user authentication.

TABLE 16–8 Broker Security Properties for LDAP Authentication

Property	Type	Default Value	Description
imq.user_repository.ldap.server	String	None	Host name and port number for LDAP server The value is of the form <i>hostName:port</i> where <i>hostName</i> is the fully qualified DNS name of the host running the LDAP server and <i>port</i> is the port number used by the server.

TABLE 16–8 Broker Security Properties for LDAP Authentication (Continued)

Property	Type	Default Value	Description
			<p>To specify a list of failover servers, use the following syntax:</p> <pre>host1:port1 ldap://host2:port2 ldap://host3:port3 ...</pre> <p>Entries in the list are separated by spaces. Note that each failover server address is prefixed with <code>ldap://</code>. Use this format even if you use SSL and have set the property <code>imq.user_repository.ldap.ssl.enabled</code> to <code>true</code>. You need not specify <code>ldaps</code> in the address.</p>
<code>imq.user_repository.ldap.principal</code>	String	None	<p>Distinguished name for binding to LDAP user repository</p> <p>Not needed if the LDAP server allows anonymous searches.</p>
<code>imq.user_repository.ldap.password</code> ¹	String	None	<p>Password for binding to LDAP user repository</p> <p>Not needed if the LDAP server allows anonymous searches.</p>
<code>imq.user_repository.ldap.propertyName</code>			
<code>imq.user_repository.ldap.base</code>	String	None	Directory base for LDAP user entries
<code>imq.user_repository.ldap.uidattr</code>	String	None	Provider-specific attribute identifier for LDAP user name

¹ Should be used only in password files

TABLE 16–8 Broker Security Properties for LDAP Authentication (Continued)

Property	Type	Default Value	Description
<code>imq.user_repository.ldap.usrformat</code>	String	None	When set to a value of <code>dn</code> , specifies that DN username format is used for authentication (for example: <code>uid=mquser,ou=People,dc=red,dc=sun,dc=com</code>). Also, the broker extracts the value of the <code>imq.user_repository.ldap.uidattr</code> attribute from the DN username, and uses this value as the user name in access control operations. If not set, then normal username format is used.
<code>imq.user_repository.ldap.usrfilter</code> ²	String	None	JNDI filter for LDAP user searches
<code>imq.user_repository.ldap.grpsearch</code>	Boolean	<code>false</code>	Enable LDAP group searches? Note – Message Queue does not support nested groups.
<code>imq.user_repository.ldap.grpbase</code>	String	None	Directory base for LDAP group entries
<code>imq.user_repository.ldap.gidattr</code>	String	None	Provider-specific attribute identifier for LDAP group name
<code>imq.user_repository.ldap.memattr</code>	String	None	Provider-specific attribute identifier for user names in LDAP group
<code>imq.user_repository.ldap.grpfilter</code> ²	String	None	JNDI filter for LDAP group searches
<code>imq.user_repository.ldap.timeout</code>	Integer	280	Time limit for LDAP searches, in seconds
<code>imq.user_repository.ldap.ssl.enabled</code>	Boolean	<code>false</code>	Use SSL when communicating with LDAP server?

² Optional

Table 16–9 lists broker properties related to JAAS-based user authentication.

TABLE 16–9 Broker Security Properties for JAAS Authentication

Property	Type	Default Value	Description
imq.user_repository.jaas.name	String	None	Set to the name of the desired entry (in the JAAS configuration file) that references the login modules you want to use as the authentication service. This is the name you noted in Step 3.
imq.user_repository.jaas.userPrincipalClass	String	None	This property, used by Message Queue access control, specifies the <code>java.security.Principal</code> implementation class in the login module(s) that the broker uses to extract the Principal name to represent the user entity in the Message Queue access control file. If, it is not specified, the user name passed from the Message Queue client when a connection was requested is used instead.
imq.user_repository.jaas.groupPrincipalClass	String	None	This property, used by Message Queue access control, specifies the <code>java.security.Principal</code> implementation class in the login module(s) that the broker uses to extract the Principal name to represent the group entity in the Message Queue access control file. If, it is not specified, the user name passed from the Message Queue client when a connection was requested is used instead.

Monitoring Properties

Table 16–10 lists the broker properties related to monitoring services.

TABLE 16-10 Broker Monitoring Properties

Property	Type	Default Value	Description
<code>imq.log.level</code> ¹	String	INFO	<p>Logging level</p> <p>Specifies the categories of logging information that can be written to an output channel. Possible values, from high to low:</p> <ul style="list-style-type: none"> ERROR WARNING INFO <p>Each level includes those above it (for example, WARNING includes ERROR).</p>
<code>imq.destination.logDeadMsgs</code> ¹	Boolean	false	<p>Log information about dead messages?</p> <p>If true, the following events will be logged:</p> <ul style="list-style-type: none"> ■ A destination is full, having reached its maximum size or message count. ■ The broker discards a message for a reason other than an administrative command or delivery acknowledgment. ■ The broker moves a message to the dead message queue.
<code>imq.log.console.stream</code>	String	ERR	<p>Destination for console output:</p> <ul style="list-style-type: none"> OUT: stdout ERR: stderr

¹ Can be used with `imqcmd update bkr` command

TABLE 16–10 Broker Monitoring Properties (Continued)

Property	Type	Default Value	Description
<code>imq.log.console.output</code>	String	ERROR WARNING	<p>Categories of logging information to write to console:</p> <p>NONE ERROR WARNING INFO ALL</p> <p>The ERROR, WARNING, and INFO categories do <i>not</i> include those above them, so each must be specified explicitly if desired. Any combination of categories can be specified, separated by vertical bars (<code> </code>).</p>
<code>imq.log.file.dirpath</code>	String	See Appendix A, “Platform-Specific Locations of Message Queue Data”	Path to directory containing log file
<code>imq.log.file.filename</code>	String	<code>log.txt</code>	Name of log file
<code>imq.log.file.output</code>	String	ALL	<p>Categories of logging information to write to log file:</p> <p>NONE ERROR WARNING INFO ALL</p> <p>The ERROR, WARNING, and INFO categories do <i>not</i> include those above them, so each must be specified explicitly if desired. Any combination of categories can be specified, separated by vertical bars (<code> </code>).</p>
<code>imq.log.file.rolloverbytes¹</code>	Integer	–1	<p>File length, in bytes, at which output rolls over to a new log file</p> <p>A value of –1 denotes an unlimited number of bytes (no rollover based on file length).</p>

¹ Can be used with `imqcmd update bkr` command

TABLE 16-10 Broker Monitoring Properties (Continued)

Property	Type	Default Value	Description
<code>imq.log.file.rolloversecs</code> ¹	Integer	604800 (one week)	Age of file, in seconds, at which output rolls over to a new log file A value of -1 denotes an unlimited number of seconds (no rollover based on file age).
<code>imq.log.syslog.output</code> ²	String	ERROR	Categories of logging information to write to <code>syslogd (1M)</code> : NONE ERROR WARNING INFO ALL The ERROR, WARNING, and INFO categories do <i>not</i> include those above them, so each must be specified explicitly if desired. Any combination of categories can be specified, separated by vertical bars (<code> </code>).
<code>imq.log.syslog.facility</code> ²	String	LOG_DAEMON	<code>syslog</code> facility for logging messages Possible values mirror those listed on the <code>syslog (3C)</code> man page. Appropriate values for use with Message Queue include: LOG_USER LOG_DAEMON LOG_LOCAL0 LOG_LOCAL1 LOG_LOCAL2 LOG_LOCAL3 LOG_LOCAL4 LOG_LOCAL5 LOG_LOCAL6 LOG_LOCAL7
<code>imq.log.syslog.identity</code> ²	String	<code>imqbrokerd_\${imq.instanceName}</code>	Identity string to be prefixed to all messages logged to <code>syslog</code>
<code>imq.log.syslog.logpid</code> ²	Boolean	true	Log broker process ID with message?

¹ Can be used with `imqcmd update bkr` command² Solaris platform only

TABLE 16–10 Broker Monitoring Properties (Continued)

Property	Type	Default Value	Description
<code>imq.log.syslog.logconsole</code> ²	Boolean	<code>false</code>	Write messages to system console if they cannot be sent to <code>syslog</code> ?
<code>imq.log.timezone</code>	String	Local time zone	<p>Time zone for log time stamps</p> <p>Possible values are the same as those used by the method <code>java.util.TimeZone.getTimeZone</code>.</p> <p>Examples:</p> <ul style="list-style-type: none"> GMT GMT–8:00 America/LosAngeles Europe/Rome Asia/Tokyo
<code>imq.metrics.enabled</code>	Boolean	<code>true</code>	<p>Enable writing of metrics information to Logger?</p> <p>Does not affect the production of metrics messages (controlled by <code>imq.metrics.topic.enabled</code>).</p>
<code>imq.metrics.interval</code>	Integer	<code>–1</code>	<p>Time interval, in seconds, at which to write metrics information to Logger</p> <p>Does not affect the time interval for production of metrics messages (controlled by <code>imq.metrics.topic.interval</code>).</p> <p>A value of <code>–1</code> denotes an indefinite interval (never write metrics information to Logger).</p>
<code>imq.metrics.topic.enabled</code>	Boolean	<code>true</code>	<p>Enable production of metrics messages to metric topic destinations?</p> <p>If <code>false</code>, an attempt to subscribe to a metric topic destination will throw a client-side exception.</p>
<code>imq.metrics.topic.interval</code>	Integer	<code>60</code>	Time interval, in seconds, at which to produce metrics messages to metric topic destinations
<code>imq.metrics.topic.persist</code>	Boolean	<code>false</code>	Are metrics messages sent to metric topic destinations persistent?

² Solaris platform only

TABLE 16–10 Broker Monitoring Properties (Continued)

Property	Type	Default Value	Description
<code>imq.metrics.topic.timetolive</code>	Integer	300	Lifetime, in seconds, of metrics messages sent to metric topic destinations
<code>imq.primaryowner.name</code> ³	String	System property <code>user.name</code> (user who started the broker)	Name of primary system owner
<code>imq.primaryowner.contact</code> ³	String	System property <code>user.name</code> (user who started the broker)	Contact information for primary system owner
<code>imq.broker.adminDefinedRoles.count</code> ³	Integer	None	Number of defined roles
<code>imq.broker.adminDefinedRoles.nameN</code> ³	String	Broker instance name	Name of defined role <i>N</i> (where <i>N</i> ranges from 0 to <code>.count-1</code>) Example: ...name0=Stocks JMS Server ...name1=JMS provider for appserver

³ Used by JES Monitoring Framework

Cluster Configuration Properties

Table 16–11 lists the configuration properties related to broker clusters.

TABLE 16–11 Broker Properties for Cluster Configuration

Property	Type	Default Value	Description
<code>imq.cluster.url</code> ^{1,2}	String	None	URL of cluster configuration file, if any Examples: <code>http://webserver/imq/cluster.properties</code> (for a file on a Web server) <code>file:/net/mfsserver/imq/cluster.properties</code> (for a file on a shared drive)
<code>imq.cluster.ha</code>	Boolean	false	Is broker part of a high-availability cluster?

¹ Must have the same value for all brokers in a cluster² Can be used with `imqcmd update bkr` command

TABLE 16–11 Broker Properties for Cluster Configuration (Continued)

Property	Type	Default Value	Description
imq.cluster.brokerlist ^{1,3}	String	None	<p>List of broker addresses belonging to cluster</p> <p>The list consists of one or more addresses, separated by commas. Each address specifies the host name and Port Mapper port number of a broker in the cluster, in the form <i>hostName:portNumber</i>.</p> <p>Example:</p> <p>host1:3000,host2:8000,ctrlhost</p> <p>Note – If set, this property is ignored (and a warning logged) for high-availability clusters; all brokers configured to use the cluster’s shared persistent store are automatically recognized as members of the cluster.</p>
imq.cluster.hostname ⁴	String	None	<p>Host name or IP address for cluster connection service</p> <p>If specified, overrides imq.hostname (see Table 16–1) for the cluster connection service. This might be necessary, for instance, if the broker’s host computer has more than one interface card installed.</p>
imq.cluster.port ⁴	Integer	0	<p>Port number for cluster connection service</p> <p>A value of 0 specifies that the port number should be allocated dynamically by the Port Mapper. You might need to set a different value, for instance, to specify a static port number for connecting to the broker through a firewall.</p>
imq.cluster.transport ¹	String	tcp	<p>Network transport protocol for cluster connection service</p> <p>For secure, encrypted message delivery between brokers, set this property to ssl.</p>

¹ Must have the same value for all brokers in a cluster

³ Conventional clusters only

⁴ Can be specified independently for each broker in a cluster

TABLE 16–11 Broker Properties for Cluster Configuration (Continued)

Property	Type	Default Value	Description
<code>imq.cluster.masterbroker</code> ^{3,1}	String	None	<p>Host name and Port Mapper port number of host on which cluster's master broker (if any) is running</p> <p>The value has the form <i>hostName:portNumber</i>, where <i>hostName</i> is the host name of the master broker's host and <i>portNumber</i> is its Port Mapper port number.</p> <p>Example:</p> <pre>ctrlhost:7676</pre> <p>Note – High-availability clusters cannot have a master broker. If this property is set for a broker belonging to a high-availability cluster, the broker will log a warning message and ignore the property.</p>
<code>imq.cluster.clusterid</code> ^{5,1}	String	None	<p>Cluster identifier</p> <p>Must be a unique alphanumeric string of no more than <i>n</i>–13 characters, where <i>n</i> is the maximum table name length allowed by the database. No two running clusters may have the same cluster identifier.</p> <p>This string is appended to the names of all database tables in the cluster's shared persistent store.</p> <p>Note – For brokers belonging to a high-availability cluster, this property is used in database table names in place of <code>imq.brokerid</code> (see Table 16–1).</p>
<code>imq.cluster.heartbeat.hostname</code> ⁵	String	None	<p>Host name or IP address for heartbeat service</p> <p>If specified, overrides <code>imq.hostname</code> (see Table 16–1) for the heartbeat service.</p>
<code>imq.cluster.heartbeat.port</code> ⁵	Integer	7676	<p>Port number for heartbeat service</p> <p>A value of 0 specifies that the port number should be allocated dynamically by the Port Mapper.</p>
<code>imq.cluster.heartbeat.interval</code> ⁵	Integer	2	Interval between heartbeats, in seconds
<code>imq.cluster.heartbeat.threshold</code> ⁵	Integer	3	Number of missed heartbeat intervals after which to invoke monitor service

³ Conventional clusters only¹ Must have the same value for all brokers in a cluster⁵ high-availability clusters only

TABLE 16–11 Broker Properties for Cluster Configuration (Continued)

Property	Type	Default Value	Description
imq.cluster.monitor.interval ⁵	Integer	30	Interval, in seconds, at which to update monitor time stamp Note – Larger values for this property will reduce the frequency of database access and thus improve overall system performance, but at the cost of slower detection and takeover in the event of broker failure.
imq.cluster.monitor.threshold ⁵	Integer	2	Number of missed monitor intervals after which to initiate broker takeover

⁵ high-availability clusters only

JMX Properties

The broker properties listed in [Table 16–12](#) support the use of the Java Management Extensions (JMX) application programming interface by Java applications. The JMX API is used to configure and monitor broker resources.

These JMX-related properties can be set in the broker's instance configuration file (`config.properties`) or at broker startup with the `-D` option of the Broker utility (`imqbrokerd`). None of these properties can be set dynamically with the Command utility (`imqcmd`).

In addition, some of these properties (`imq.jmx.rmiregistry.start`, `imq.jmx.rmiregistry.use`, `imq.jmx.rmiregistry.port`) can be set with corresponding Broker utility `imqbrokerd` options described in [Table 15–1](#).

See [Appendix D, “JMX Support,”](#) for further information on administrative support of JMX clients.

TABLE 16–12 Broker Properties for JMX Support

Property	Type	Default Value	Description
imq.jmx.connector.activelist	String	jmxrmi	Names of JMX connectors to be activated at broker startup, separated by commas

TABLE 16–12 Broker Properties for JMX Support (Continued)

Property	Type	Default Value	Description
<code>imq.jmx.connector.RMIconnectorName.urlpath</code>	String	Shown in next column	<p><code>urlpath</code> component of JMX service URL for connector <code>connectorName</code></p> <p>Useful in cases where an RMI registry is being used and the JMX service URL path must be set explicitly (such as when a shared external RMI registry is used). See “The JMX Service URL” on page 400.</p> <p>Default:</p> <pre>/jndi/rmi://brokerHost:rmiPort /brokerHost/brokerPort/connectorName</pre>
<code>imq.jmx.connector.RMIconnectorName.port</code>	Integer	None: the port is dynamically allocated	<p>Port number of JMX connector</p> <p>Used to specify a static/known JMX connector port, typically in cases where a JMX client is accessing the broker's MBean server through a firewall. See “JMX Connections Through a Firewall” on page 407.</p>
<code>imq.jmx.connector.RMIconnectorName.useSSL</code>	Boolean	false	<p>Use Secure Socket Layer (SSL) for connector <code>connectorName</code>?</p> <p>This property is set to <code>true</code> for the <code>ssljmxrmi</code> connector.</p>
<code>imq.jmx.connector.RMIconnectorName.brokerHostTrusted</code>	Boolean	false	<p>Trust any certificate presented by broker for connector <code>connectorName</code>?</p> <p>Applies only when <code>imq.jmx.connector.connectorName.useSSL</code> is <code>true</code>.</p> <p>If <code>false</code>, the JMX client runtime will validate all certificates presented to it. Validation will fail if the signer of the certificate is not in the client's trust store.</p> <p>If <code>true</code>, validation of certificates is skipped. This can be useful, for instance, during software testing when a self-signed certificate is used.</p>

TABLE 16–12 Broker Properties for JMX Support (Continued)

Property	Type	Default Value	Description
imq.jmx.rmiregistry.start	Boolean	false	<p>Start RMI registry at broker startup?</p> <p>If <code>true</code>, the broker will start an RMI registry at the port specified by <code>imq.jmx.rmiregistry.port</code> and use the registry to store the JMX connector stub. (The value of <code>imq.jmx.rmiregistry.use</code> is ignored in this case.)</p> <p>For convenience, this property can also be set at broker startup with the <code>-startRmiRegistry</code> option of <code>imqbrokerd</code>.</p>
imq.jmx.rmiregistry.use	Boolean	false	<p>Use an existing RMI registry?</p> <p>Applies only if <code>imq.jmx.rmiregistry.start</code> is <code>false</code>.</p> <p>If <code>true</code>, the broker will use an existing RMI registry on the local host at the port specified by <code>imq.jmx.rmiregistry.port</code> to store the JMX connector stub. The existing RMI registry must already be running at broker startup.</p> <p>For convenience, this property can also be set at broker startup with the <code>-useRmiRegistry</code> option of <code>imqbrokerd</code>.</p>
imq.jmx.rmiregistry.port	Integer	1099	<p>Port number of RMI registry</p> <p>Applies only if <code>imq.jmx.rmiregistry.start</code> is <code>true</code> or <code>imq.jmx.rmiregistry.use</code> is <code>true</code>.</p> <p>This port number will be included in the URL path of the JMX service URL.</p> <p>For convenience, this property can also be set at broker startup with the <code>-rmiRegistryPort</code> option of <code>imqbrokerd</code>.</p>

Alphabetical List of Broker Properties

“[Alphabetical List of Broker Properties](#)” on page 326 is an alphabetical list of broker configuration properties, with cross-references to the relevant tables in this chapter.

TABLE 16-13 Alphabetical List of Broker Properties

Property	Table
<code>imq.accesscontrol.enabled</code>	Table 16-7
<code>imq.accesscontrol.type</code>	Table 16-7
<code>imq.accesscontrol.file.filename</code>	Table 16-7
<code>imq.authentication.basic.user_repository</code>	Table 16-7
<code>imq.authentication.client.response.timeout</code>	Table 16-7
<code>imq.authentication.type</code>	Table 16-7
<code>imq.autocreate.destination.isLocalOnly</code>	Table 16-3
<code>imq.autocreate.destination.limitBehavior</code>	Table 16-3
<code>imq.autocreate.destination.maxBytesPerMsg</code>	Table 16-3
<code>imq.autocreate.destination.maxNumMsgs</code>	Table 16-3
<code>imq.autocreate.destination.maxNumProducers</code>	Table 16-3
<code>imq.autocreate.destination.maxTotalMsgBytes</code>	Table 16-3
<code>imq.autocreate.destination.useDMQ</code>	Table 16-3
<code>imq.autocreate.queue</code>	Table 16-3
<code>imq.autocreate.queue.consumerFlowLimit</code>	Table 16-3
<code>imq.autocreate.queue.localDeliveryPreferred</code>	Table 16-3
<code>imq.autocreate.queue.maxNumActiveConsumers</code>	Table 16-3
<code>imq.autocreate.queue.maxNumBackupConsumers</code>	Table 16-3
<code>imq.autocreate.topic</code>	Table 16-3
<code>imq.autocreate.topic.consumerFlowLimit</code>	Table 16-3
<code>imq.broker.adminDefinedRoles.count</code>	Table 16-10
<code>imq.broker.adminDefinedRoles.names</code>	Table 16-10
<code>imq.brokerid</code>	Table 16-1
<code>imq.cluster.brokerlist</code>	Table 16-11
<code>imq.cluster.clusterid</code>	Table 16-11
<code>imq.cluster.ha</code>	Table 16-11
<code>imq.cluster.heartbeat.hostname</code>	Table 16-11

TABLE 16-13 Alphabetical List of Broker Properties (Continued)

Property	Table
<code>imq.cluster.heartbeat.interval</code>	Table 16-11
<code>imq.cluster.heartbeat.port</code>	Table 16-11
<code>imq.cluster.heartbeat.threshold</code>	Table 16-11
<code>imq.cluster.hostname</code>	Table 16-11
<code>imq.cluster.masterbroker</code>	Table 16-11
<code>imq.cluster.monitor.interval</code>	Table 16-11
<code>imq.cluster.monitor.threshold</code>	Table 16-11
<code>imq.cluster.port</code>	Table 16-11
<code>imq.cluster.transport</code>	Table 16-11
<code>imq.cluster.url</code>	Table 16-11
<code>imq.destination.DMQ.truncateBody</code>	Table 16-2
<code>imq.destination.logDeadMsgs</code>	Table 16-10
<code>imq.hostname</code>	Table 16-1
<code>imq.imqcmd.password</code>	Table 16-7
<code>imq.jmx.connector.activelist</code>	Table 16-12
<code>imq.jmx.connector.RMIconnectorName.brokerHostTrusted</code>	Table 16-12
<code>imq.jmx.connector.RMIconnectorName.port</code>	Table 16-12
<code>imq.jmx.connector.RMIconnectorName.urlpath</code>	Table 16-12
<code>imq.jmx.connector.RMIconnectorName.useSSL</code>	Table 16-12
<code>imq.jmx.rmiregistry.port</code>	Table 16-12
<code>imq.jmx.rmiregistry.start</code>	Table 16-12
<code>imq.jmx.rmiregistry.use</code>	Table 16-12
<code>imq.keystore.file.dirpath</code>	Table 16-7
<code>imq.keystore.file.name</code>	Table 16-7
<code>imq.keystore.password</code>	Table 16-7
<code>imq.keystore.propertyName</code>	Table 16-7
<code>imq.log.console.output</code>	Table 16-10
<code>imq.log.console.stream</code>	Table 16-10

TABLE 16-13 Alphabetical List of Broker Properties (Continued)

Property	Table
<code>imq.log.file.dirpath</code>	Table 16-10
<code>imq.log.file.filename</code>	Table 16-10
<code>imq.log.file.output</code>	Table 16-10
<code>imq.log.file.rolloverbytes</code>	Table 16-10
<code>imq.log.file.rolloversecs</code>	Table 16-10
<code>imq.log.level</code>	Table 16-10
<code>imq.log.syslog.facility</code>	Table 16-10
<code>imq.log.syslog.identity</code>	Table 16-10
<code>imq.log.syslog.logconsole</code>	Table 16-10
<code>imq.log.syslog.logpid</code>	Table 16-10
<code>imq.log.syslog.output</code>	Table 16-10
<code>imq.log.timezone</code>	Table 16-10
<code>imq.message.expiration.interval</code>	Table 16-2
<code>imq.message.max_size</code>	Table 16-2
<code>imq.metrics.enabled</code>	Table 16-10
<code>imq.metrics.interval</code>	Table 16-10
<code>imq.metrics.topic.enabled</code>	Table 16-10
<code>imq.metrics.topic.interval</code>	Table 16-10
<code>imq.metrics.topic.persist</code>	Table 16-10
<code>imq.metrics.topic.timetolive</code>	Table 16-10
<code>imq.passfile.dirpath</code>	Table 16-7
<code>imq.passfile.enabled</code>	Table 16-7
<code>imq.passfile.name</code>	Table 16-7
<code>imq.persist.file.destination.message.filepool.limit</code>	Table 16-5
<code>imq.persist.file.message.cleanup</code>	Table 16-5
<code>imq.persist.file.message.filepool.cleanratio</code>	Table 16-5
<code>imq.persist.file.message.max_record_size</code>	Table 16-5
<code>imq.persist.file.sync.enabled</code>	Table 16-5

TABLE 16-13 Alphabetical List of Broker Properties (Continued)

Property	Table
<code>imq.persist.file.transaction.memorymappedfile.enabled</code>	Table 16-5
<code>imq.persist.jdbc.dbVendor</code>	Table 16-6
<code>imq.persist.jdbc.vendorName.closedburl</code>	Table 16-6
<code>imq.persist.jdbc.vendorName.createdburl</code>	Table 16-6
<code>imq.persist.jdbc.vendorName.driver</code>	Table 16-6
<code>imq.persist.jdbc.vendorName.needpassword</code>	Table 16-6
<code>imq.persist.jdbc.vendorName.opendburl</code>	Table 16-6
<code>imq.persist.jdbc.vendorName.password</code>	Table 16-6
<code>imq.persist.jdbc.vendorName.property.propName</code>	Table 16-6
<code>imq.persist.jdbc.vendorName.user</code>	Table 16-6
<code>imq.persist.store</code>	Table 16-4
<code>imq.ping.interval</code>	Table 16-1
<code>imq.portmapper.backlog</code>	Table 16-1
<code>imq.portmapper.hostname</code>	Table 16-1
<code>imq.portmapper.port</code>	Table 16-1
<code>imq.primaryowner.contact</code>	Table 16-10
<code>imq.primaryowner.name</code>	Table 16-10
<code>imq.resourceState.count</code>	Table 16-2
<code>imq.resourceState.threshold</code>	Table 16-2
<code>imq.service.activelist</code>	Table 16-1
<code>imq.serviceName.accesscontrol.enabled</code>	Table 16-7
<code>imq.serviceName.accesscontrol.file.filename</code>	Table 16-7
<code>imq.serviceName.authentication.type</code>	Table 16-7
<code>imq.serviceName.max_threads</code>	Table 16-1
<code>imq.serviceName.min_threads</code>	Table 16-1
<code>imq.serviceName.protocolType.hostname</code>	Table 16-1
<code>imq.serviceName.protocolType.port</code>	Table 16-1
<code>imq.serviceName.threadpool_model</code>	Table 16-1

TABLE 16-13 Alphabetical List of Broker Properties (Continued)

Property	Table
<code>imq.shared.connectionMonitor_limit</code>	Table 16-1
<code>imq.system.max_count</code>	Table 16-2
<code>imq.system.max_size</code>	Table 16-2
<code>imq.transaction.autorollback</code>	Table 16-2
<code>imq.user_repository.ldap.base</code>	Table 16-8
<code>imq.user_repository.ldap.gidattr</code>	Table 16-8
<code>imq.user_repository.ldap.grpbase</code>	Table 16-8
<code>imq.user_repository.ldap.grpfilter</code>	Table 16-8
<code>imq.user_repository.ldap.grpsearch</code>	Table 16-8
<code>imq.user_repository.ldap.memattr</code>	Table 16-8
<code>imq.user_repository.ldap.password</code>	Table 16-8
<code>imq.user_repository.ldap.principal</code>	Table 16-8
<code>imq.user_repository.ldap.propertyName</code>	Table 16-8
<code>imq.user_repository.ldap.server</code>	Table 16-8
<code>imq.user_repository.ldap.ssl.enabled</code>	Table 16-8
<code>imq.user_repository.ldap.timeout</code>	Table 16-8
<code>imq.user_repository.ldap.uidattr</code>	Table 16-8
<code>imq.user_repository.ldap.usrfilter</code>	Table 16-8
<code>imq.user_repository.jaas.name</code>	Table 16-9
<code>imq.user_repository.jaas.userPrincipalClass</code>	Table 16-9
<code>imq.user_repository.jaas.groupPrincipalClass</code>	Table 16-9

Physical Destination Property Reference

This chapter provides reference information about configuration properties for physical destinations.

Physical Destination Properties

[Table 17–1](#) lists the configuration properties for physical destinations. These properties can be set when creating or updating a physical destination. For auto-created destinations, you set default values in the broker’s instance configuration file (see [Table 16–3](#)).

TABLE 17–1 Physical Destination Properties

Property	Type	Default Value	Description
<code>maxNumMsgs</code> ¹	Integer	–1	<p>Maximum number of unconsumed messages</p> <p>A value of –1 denotes an unlimited number of messages.</p> <p>For the dead message queue, the default value is 1000.</p> <p>Note – When flow control is in effect (<code>limitBehavior = FLOW_CONTROL</code>), it is possible for the specified message limit to be exceeded because the broker cannot react quickly enough to stop the flow of incoming messages. In such cases, the value specified for <code>maxNumMsgs</code> serves as merely a hint for the broker rather than a strictly enforced limit.</p>
<code>maxBytesPerMsg</code>	String	–1	<p>Maximum size, in bytes, of any single message</p> <p>Rejection of a persistent message is reported to the producing client with an exception; no notification is sent for nonpersistent messages.</p>

¹ In a cluster environment, applies to each individual instance of a destination rather than collectively to all instances in the cluster

TABLE 17-1 Physical Destination Properties (Continued)

Property	Type	Default Value	Description
			<p>The value may be expressed in bytes, kilobytes, or megabytes, using the following suffixes:</p> <ul style="list-style-type: none"> b: Bytes k: Kilobytes (1024 bytes) m: Megabytes (1024 × 1024 = 1,048,576 bytes) <p>An unsuffixed value is expressed in bytes; a value of –1 denotes an unlimited message size.</p> <p>Examples:</p> <ul style="list-style-type: none"> 1600: 1600 bytes 1600b: 1600 bytes 16k: 16 kilobytes (= 16,384 bytes) 16m: 16 megabytes (= 16,777,216 bytes) –1: No limit
maxTotalMsgBytes ¹	String	–1	<p>Maximum total memory, in bytes, for unconsumed messages</p> <p>The syntax is the same as for maxBytesPerMsg (see above).</p> <p>For the dead message queue, the default value is 10m.</p>
limitBehavior	String	REJECT_NEWEST	<p>Broker behavior when memory-limit threshold reached:</p> <ul style="list-style-type: none"> FLOW_CONTROL: Slow down producers REMOVE_OLDEST: Throw out oldest messages REMOVE_LOW_PRIORITY: Throw out lowest-priority messages according to age; no notification to producing client REJECT_NEWEST: Reject newest messages; notify producing client with an exception only if message is persistent <p>If the value is REMOVE_OLDEST or REMOVE_LOW_PRIORITY and the useDMQ property is true, excess messages are moved to the dead message queue. For the dead message queue itself, the default limit behavior is REMOVE_OLDEST and cannot be set to FLOW_CONTROL.</p>
maxNumProducers ²	Integer	100	<p>Maximum number of message producers for destination</p> <p>When this limit is reached, no new producers can be created. A value of –1 denotes an unlimited number of producers.</p>

¹ In a cluster environment, applies to each individual instance of a destination rather than collectively to all instances in the cluster² Does not apply to dead message queue

TABLE 17-1 Physical Destination Properties (Continued)

Property	Type	Default Value	Description
<code>maxNumActiveConsumers</code> ³	Integer	-1	Maximum number of active message consumers in load-balanced delivery from queue destination A value of -1 denotes an unlimited number of consumers. This property used mostly in cases where message order is important and you want to provide backup consumers in case the principal consumer of a queue fails. If message order is not important, then you would simply use multiple consumers to provide for scalability and availability.
<code>maxNumBackupConsumers</code> ³	Integer	0	Maximum number of backup message consumers in load-balanced delivery from queue destination A value of -1 denotes an unlimited number of consumers.
<code>consumerFlowLimit</code>	Integer	1000	Maximum number of messages delivered to consumer(s) in a single batch In load-balanced queue delivery, this is the initial number of queued messages routed to active consumers before load balancing begins. The client runtime can override this limit by specifying a lower value on the connection factory object.. A value of -1 denotes an unlimited number of messages.
<code>isLocalOnly</code> ²	Boolean	false	Local delivery only? This property applies only to destinations in broker clusters, and cannot be changed once the destination has been created. If <code>true</code> , the destination is not replicated on other brokers and is limited to delivering messages only to local consumers (those connected to the broker on which the destination is created).
<code>localDeliveryPreferred</code> ^{2,3}	Boolean	false	Local delivery preferred? This property applies only to load-balanced queue delivery in broker clusters. If <code>true</code> , messages will be delivered to remote consumers only if there are no consumers on the local broker; the destination must not be restricted to local-only delivery (<code>isLocalOnly</code> must be <code>false</code>).
<code>useDMQ</code> ²	Boolean	true	Send dead messages to dead message queue? If <code>false</code> , dead messages will simply be discarded.

³ Queue destinations only² Does not apply to dead message queue

TABLE 17-1 Physical Destination Properties (Continued)

Property	Type	Default Value	Description
<code>validateXMLSchemaEnabled</code> ⁴	Boolean	<code>false</code>	<p>XML schema validation is enabled?</p> <p>When XML validation is enabled, the Message Queue client runtime will attempt to validate an XML message against the specified XSDs (or against the DTD, if no XSD is specified) before sending it to the broker. If the specified schema cannot be located or the message cannot be validated, the message is not sent, and an exception is thrown. Client applications using this feature should use JRE 1.5 or above.</p> <p>If set to <code>false</code> or not set, then XML schema validation is not enabled for the destination.</p>
<code>XMLSchemaURIList</code> ⁴	String	<code>null</code>	<p>Space separated list of XML schema document (XSD) URI strings</p> <p>The URIs point to the location of one or more XSDs to use for XML schema validation, if enabled.</p> <p>Use double quotes around this value if multiple URIs are specified.</p> <p>Example:</p> <p><code>"http://foo/flap.xsd http://test.com/test.xsd"</code></p> <p>If this property is not set or null and XML validation is enabled, XML validation is performed using a DTD specified in the XML document.</p> <p>if an XSD is changed, as a result of changing application requirements, all client applications producing XML messages based on the changed XSD must reconnect to the broker.</p>
<code>reloadXMLSchemaOnFailure</code> ⁴	Boolean	<code>false</code>	<p>Reload XML schema on failure enabled?</p> <p>If set to <code>true</code> and XML validation fails, then the Message Queue client runtime will attempt to reload the XSD before attempting again to validate a message. The client runtime will throw an exception if the validation fails using the reloaded XSD.</p> <p>If set to <code>false</code> or not set, then the schema is not reloaded if validation fails.</p>

⁴ This property should be set when a destination is inactive: when it has no consumers or producers and when there are no messages in the destination. Otherwise the producer must reconnect.

Administered Object Attribute Reference

This chapter provides reference information about the attributes of administered objects. It consists of the following sections:

- [“Connection Factory Attributes” on page 337](#)
- [“Destination Attributes” on page 345](#)

Connection Factory Attributes

The attributes of a connection factory object are grouped into categories described in the following sections below:

- [“Connection Handling” on page 337](#)
- [“Client Identification” on page 341](#)
- [“Reliability and Flow Control” on page 341](#)
- [“Queue Browser and Server Sessions” on page 343](#)
- [“Standard Message Properties” on page 344](#)
- [“Message Header Overrides” on page 344](#)

Connection Handling

[Table 18–1](#) lists the connection factory attributes for connection handling.

TABLE 18-1 Connection Factory Attributes for Connection Handling

Attribute	Type	Default Value	Description
<code>imqAddressList</code>	String	An existing Message Queue 3.0 address, if any; if none, the first entry in Table 18-2	<p>List of broker addresses</p> <p>The list consists of one or more addresses, separated by commas. Each address specifies (or implies) the host name, port number, and connection service for a broker instance to which the client can connect. Address syntax varies depending on the connection service and port assignment method; see below for details.</p> <p>Note – In a high-availability broker cluster, the value of this attribute is updated dynamically as brokers enter and leave the cluster, so that it always reflects the cluster's current membership.</p>
<code>imqAddressListBehavior</code>	String	PRIORITY	<p>Order in which to attempt connection to broker addresses:</p> <p>PRIORITY: Order specified in address list</p> <p>RANDOM: Random order</p> <p>Note – If many clients share the same connection factory, specify random connection order to prevent them from all attempting to connect to the same address.</p>
<code>imqAddressListIterations</code>	Integer	1	<p>Number of times to iterate through address list attempting to establish or reestablish a connection</p> <p>A value of –1 denotes an unlimited number of iterations.</p> <p>Note – In the event of broker failure in a high-availability broker cluster, this attribute is ignored and the Message Queue client runtime iterates through the address list indefinitely until it succeeds in reconnecting to a takeover broker. The effect is equivalent to an <code>imqAddressListIterations</code> value of –1, overriding any other explicit or default setting of this attribute. The only way for a client application to avoid this behavior is to close the connection explicitly on broker failure.</p>
<code>imqPingInterval</code>	Integer	30	<p>Interval, in seconds, at which to test connection between client and broker</p> <p>A value of 0 or –1 disables periodic testing of the connection.</p>

TABLE 18–1 Connection Factory Attributes for Connection Handling (Continued)

Attribute	Type	Default Value	Description
<code>imqReconnectEnabled</code>	Boolean	<code>false</code>	<p>Attempt to reestablish a lost connection?</p> <p>Note – In the event of broker failure in a high-availability broker cluster, this attribute is ignored and automatic reconnection is always attempted. The effect is equivalent to an <code>imqReconnectEnabled</code> value of <code>true</code>, overriding any other explicit or default setting of this attribute. The only way for a client application to avoid this behavior is to close the connection explicitly on broker failure.</p>
<code>imqReconnectAttempts</code>	Integer	<code>0</code>	<p>Number of times to attempt connection (or reconnection) to each address in address list before moving on to next</p> <p>A value of <code>-1</code> denotes an unlimited number of connection attempts: attempt repeatedly to connect to first address until successful. For example, in a high-availability broker cluster, this value will allow for connection to the failover broker.</p>
<code>imqReconnectInterval</code>	Long integer	<code>3000</code>	<p>Interval, in milliseconds, between reconnection attempts</p> <p>This value applies both for successive attempts on a given address and for successive addresses in the list.</p> <p>Note – Too small a value may give the broker insufficient recovery time; too large a value may cause unacceptable connection delays.</p>
<code>imqSSLIsHostTrusted</code>	Boolean	<code>false</code>	<p>Trust any certificate presented by broker?</p> <p>If <code>false</code>, the Message Queue client runtime will validate all certificates presented to it. Validation will fail if the signer of the certificate is not in the client's trust store.</p> <p>If <code>true</code>, validation of certificates is skipped. This can be useful, for instance, during software testing when a self-signed certificate is used.</p> <p>NOTE: To use signed certificates from a certification authority, set this attribute to <code>false</code>.</p>

The value of the `imqAddressList` attribute is a comma-separated string specifying one or more broker addresses to which to connect. The general syntax for each address is as follows:

scheme://*address*

where *scheme* identifies one of the addressing schemes shown in the first column of [Table 18–2](#) and *address* denotes the broker address itself. The exact syntax for specifying the address depends on the addressing scheme, as shown in the last column of the table.

TABLE 18–2 Message Broker Addressing Schemes

Scheme	Service	Syntax	Description
mq	jms or ssljms	<i>[hostName][:portNumber]/serviceName]</i>	<p>Assign port dynamically for <code>jms</code> or <code>ssljms</code> connection service</p> <p>The address list entry specifies the host name and port number for the Message Queue Port Mapper. The Port Mapper itself dynamically assigns a port to be used for the connection.</p> <p>Default values:</p> <p><i>hostName</i> = localhost <i>portNumber</i> = 7676 <i>serviceName</i> = <code>jms</code></p> <p>For the <code>ssljms</code> connection service, all variables must be specified explicitly.</p>
mqtcp	jms	<i>hostName:portNumber/jms</i>	<p>Connect to specified port using <code>jms</code> connection service</p> <p>Bypasses the Port Mapper and makes a TCP connection directly to the specified host name and port number.</p>
mqssl	ssljms	<i>hostName:portNumber/ssljms</i>	<p>Connect to specified port using <code>ssljms</code> connection service</p> <p>Bypasses the Port Mapper and makes a secure SSL connection directly to the specified host name and port number.</p>
http	httpjms	<p><i>http://hostName:portNumber/contextRoot/tunnel</i></p> <p>If multiple broker instances use the same tunnel servlet, the following syntax connects to a specific broker instance rather than a randomly selected one:</p> <p><i>http://hostName:portNumber/contextRoot/tunnel?ServerName=hostName:instanceName</i></p>	<p>Connect to specified port using <code>httpjms</code> connection service</p> <p>Makes an HTTP connection to a Message Queue tunnel servlet at the specified URL. The broker must be configured to access the HTTP tunnel servlet.</p>
https	httpsjms	<p><i>https://hostName:portNumber/contextRoot/tunnel</i></p> <p>If multiple broker instances use the same tunnel servlet, the following syntax connects to a specific broker instance rather than a randomly selected one:</p> <p><i>https://hostName:portNumber/contextRoot/tunnel?ServerName=hostName:instanceName</i></p>	<p>Connect to specified port using <code>httpsjms</code> connection service</p> <p>Makes a secure HTTPS connection to a Message Queue tunnel servlet at the specified URL. The broker must be configured to access the HTTPS tunnel servlet.</p>

TABLE 18–3 Message Broker Address Examples

Service	Broker Host	Port	Example Address
Not specified	Not specified	Not specified	No address (mq://localhost:7676/jms)
Not specified	Specified host	Not specified	myBkrHost (mq://myBkrHost:7676/jms)
Not specified	Not specified	Specified Port Mapper port	1012 (mq://localhost:1012/jms)
ssljms	Local host	Standard Port Mapper port	mq://localhost:7676/ssljms
ssljms	Specified host	Standard Port Mapper port	mq://myBkrHost:7676/ssljms
ssljms	Specified host	Specified Port Mapper port	mq://myBkrHost:1012/ssljms
jms	Local host	Specified service port	mqtcp://localhost:1032/jms
ssljms	Specified host	Specified service port	mqssl://myBkrHost:1034/ssljms
httpjms	Not applicable	Not applicable	http://websrvr1:8085/imq/tunnel
httpsjms	Not applicable	Not applicable	https://websrvr2:8090/imq/tunnel

Client Identification

Table 18–4 lists the connection factory attributes for client identification.

TABLE 18–4 Connection Factory Attributes for Client Identification

Attribute	Type	Default Value	Description
imqDefaultUsername	String	guest	Default user name for authenticating with broker
imqDefaultPassword	String	guest	Default password for authenticating with broker
imqConfiguredClientID	String	null	Administratively configured client identifier
imqDisableSetClientID	Boolean	false	Prevent client from changing client identifier using setClientID method?

Reliability and Flow Control

Table 18–5 lists the connection factory attributes for reliability and flow control.

TABLE 18–5 Connection Factory Attributes for Reliability and Flow Control

Attribute	Type	Default Value	Description
<code>imqAckTimeout</code>	String	<code>0</code>	<p>Maximum time, in milliseconds, to wait for broker acknowledgment before throwing an exception</p> <p>A value of <code>0</code> denotes no timeout (wait indefinitely).</p> <p>Note – In some situations, too low a value can cause premature timeout: for example, initial authentication of a user against an LDAP user repository using a secure (SSL) connection can take more than 30 seconds.</p>
<code>imqConnectionFlowCount</code>	Integer	<code>100</code>	<p>Number of payload messages in a metered batch</p> <p>Delivery of payload messages to the client is temporarily suspended after this number of messages, allowing any accumulated control messages to be delivered. Payload message delivery is resumed on notification by the client runtime, and continues until the count is again reached.</p> <p>A value of <code>0</code> disables metering of message delivery and may cause Message Queue control messages to be blocked by heavy payload message traffic.</p>
<code>imqConnectionFlowLimitEnabled</code>	Boolean	<code>false</code>	Limit message flow at connection level?
<code>imqConnectionFlowLimit</code>	Integer	<code>1000</code>	<p>Maximum number of messages per connection to deliver and buffer for consumption</p> <p>Message delivery on a connection stops when the number of unconsumed payload messages pending (subject to flow metering governed by <code>imqConnectionFlowCount</code>) exceeds this limit. Delivery resumes only when the number of pending messages falls below the limit. This prevents the client from being overwhelmed with pending messages that might cause it to run out of memory.</p> <p>This attribute is ignored if <code>imqConnectionFlowLimitEnabled</code> is <code>false</code>.</p>

TABLE 18–5 Connection Factory Attributes for Reliability and Flow Control (Continued)

Attribute	Type	Default Value	Description
<code>imqConsumerFlowLimit</code>	Integer	1000	<p>Maximum number of messages per consumer to deliver and buffer for consumption</p> <p>Message delivery to a given consumer stops when the number of unconsumed payload messages pending for that consumer exceeds this limit. Delivery resumes only when the number of pending messages for the consumer falls below the percentage specified by <code>imqConsumerFlowThreshold</code>. This can be used to improve load balancing among multiple consumers and prevent any single consumer from starving others on the same connection.</p> <p>This limit can be overridden by a lower value set for a queue's own <code>consumerFlowLimit</code> attribute (see Chapter 17, “Physical Destination Property Reference”). Note also that message delivery to all consumers on a connection is subject to the overall limit specified by <code>imqConnectionFlowLimit</code>.</p>
<code>imqConsumerFlowThreshold</code>	Integer	50	Number of messages per consumer buffered in the client runtime, as a percentage of <code>imqConsumerFlowLimit</code> , below which to resume message delivery

Queue Browser and Server Sessions

[Table 18–6](#) lists the connection factory attributes for queue browsing and server sessions.

TABLE 18–6 Connection Factory Attributes for Queue Browser and Server Sessions

Attribute	Type	Default Value	Description
<code>imqQueueBrowserMaxMessagesPerRetrieve</code>	Integer	1000	<p>Maximum number of messages to retrieve at one time when browsing contents of a queue destination</p> <p>Note – This attribute does not affect the total number of messages browsed, only the way they are chunked for delivery to the client runtime (fewer but larger chunks or more but smaller ones). The client application will always receive all messages in the queue. Changing the attribute's value may affect performance, but will not affect the total amount of data retrieved.</p>
<code>imqQueueBrowserRetrieveTimeout</code>	Long integer	60000	Maximum time, in milliseconds, to wait to retrieve messages, when browsing contents of a queue destination, before throwing an exception

TABLE 18–6 Connection Factory Attributes for Queue Browser and Server Sessions (Continued)

Attribute	Type	Default Value	Description
imqLoadMaxToServerSession	Boolean	true	Load up to maximum number of messages into a server session? If false, the client will load only a single message at a time. This attribute applies only to JMS application server facilities.

Standard Message Properties

The connection factory attributes listed in [Table 18–7](#) control whether the Message Queue client runtime sets certain standard message properties defined in the *Java Message Service Specification*.

TABLE 18–7 Connection Factory Attributes for Standard Message Properties

Property	Type	Default Value	Description
imqSetJMSXUserID	Boolean	false	Set JMSXUserID property (identity of user sending message) for produced messages?
imqSetJMSXAppID	Boolean	false	Set JMSXAppID property (identity of application sending message) for produced messages?
imqSetJMSXProducerTXID	Boolean	false	Set JMSXProducerTXID property (transaction identifier of transaction within which message was produced) for produced messages?
imqSetJMSXConsumerTXID	Boolean	false	Set JMSXConsumerTXID property (transaction identifier of transaction within which message was consumed) for consumed messages?
imqSetJMSXRcvTimestamp	Boolean	false	Set JMSXRcvTimestamp property (time message delivered to consumer) for consumed messages?

Message Header Overrides

[Table 18–8](#) lists the connection factory attributes for overriding JMS message header fields.

TABLE 18–8 Connection Factory Attributes for Message Header Overrides

Attribute	Type	Default Value	Description
imqOverrideJMSDeliveryMode	Boolean	false	Allow client-set delivery mode to be overridden?
imqJMSDeliveryMode	Integer	2	Overriding value of delivery mode: 1 Nonpersistent 2 Persistent
imqOverrideJMSExpiration	Boolean	false	Allow client-set expiration time to be overridden?
imqJMSExpiration	Long integer	0	Overriding value of expiration time, in milliseconds A value of 0 denotes an unlimited expiration time (message never expires).
imqOverrideJMSPriority	Boolean	false	Allow client-set priority level to be overridden?
imqJMSPriority	Integer	4 (normal)	Overriding value of priority level (0 to 9)
imqOverrideJMSHeadersToTemporaryDestinations	Boolean	false	Apply overrides to temporary destinations?

Destination Attributes

Table 18–9 lists the attributes that can be set for a destination administered object.

TABLE 18–9 Destination Attributes

Attribute	Type	Default Value	Description
imqDestinationName	String	Untitled_Destination_Object	Name of physical destination The destination name may contain only alphanumeric characters (no spaces) and must begin with an alphabetic character or the underscore (_) or dollar sign (\$) character. It may not begin with the characters mq.
imqDestinationDescription	String	None	Descriptive string for destination

JMS Resource Adapter Property Reference

This chapter describes the configuration properties of the Message Queue JMS Resource Adapter (JMS RA), which enables you to integrate Sun Java™ System Message Queue with any J2EE 1.4 application server by means of the standard J2EE connector architecture (JCA). When plugged into an application server, the Resource Adapter allows applications deployed in that application server to use Message Queue to send and receive JMS messages.

The Message Queue JMS Resource Adapter exposes its configuration properties through three JavaBean components:

- The `ResourceAdapter` JavaBean (“[ResourceAdapter JavaBean](#)” on page 347) affects the behavior of the Resource Adapter as a whole.
- The `ManagedConnectionFactory` JavaBean (“[ManagedConnectionFactory JavaBean](#)” on page 350) affects connections created by the Resource Adapter for use by message-driven beans (MDBs).
- The `ActivationSpec` JavaBean (“[ActivationSpec JavaBean](#)” on page 351) affects message endpoints that represent MDBs in their interactions with the messaging system.

To set property values for these entities, you use the tools provided by your application server for configuration and deployment of the Resource Adapter and for deployment of MDBs.

This chapter lists and describes the configuration properties of the Message Queue JMS Resource Adapter. It contains the following sections:

ResourceAdapter JavaBean

The `ResourceAdapter` configuration configures the default JMS Resource Adapter behavior. [Table 19–1](#) lists and describes the properties with which you can configure this JavaBean.

TABLE 19-1 Resource Adapter Properties

Property	Type	Default Value	Description
addressList ¹	String	mq://localhost:7676/jms	Message service address for connecting to Message Queue service Equivalent to connectionURL (below).
connectionURL ¹	String	mq://localhost:7676/jms	Message service address for connecting to the Message Queue service Equivalent to addressList(above).
brokerInstanceName	String	imqbroker	Name of broker instance
brokerPort	Integer	7676	Port number for connecting to broker
brokerBindAddress	String	Null	Address to which broker binds on host machine If null, the broker will bind to all addresses on the host machine.
userName ²	String	guest	Default user name for connecting to Message Queue service
password ²	String	guest	Default password for connecting to Message Queue service
addressListBehavior	String	PRIORITY	Order in which to attempt connection to Message Queue service: PRIORITY: Order specified in address list RANDOM: Random order Note – Reconnection attempts after a connection failure start with the broker whose connection failed and proceed sequentially through the address list, regardless of the value set for this property.
addressListIterations	Integer	1	Number of times to iterate through address list attempting to establish or reestablish a connection
reconnectEnabled	Boolean	false	Attempt to reestablish a lost connection?

¹ Exactly one of these properties must be specified² Required

TABLE 19-1 Resource Adapter Properties (Continued)

Property	Type	Default Value	Description
reconnectAttempts	Integer	6	Number of times to attempt reconnection to each address in address list before moving on to next
reconnectInterval	Long integer	30000	Interval, in milliseconds, between reconnection attempts
brokerEnableHA	Boolean	false	Enable high availability?
clusterID	String	None	<p>Cluster identifier</p> <p>If specified, only brokers with the same cluster identifier can be clustered together. In the event of broker failure, client connections will fail over only to brokers with the same cluster identifier as the original broker. If not specified, client connections can fail over to any other broker with an unspecified cluster identifier.</p> <p>For standalone brokers (those not belonging to a cluster), this property is ignored.</p> <p>The identifier may contain only alphabetic letters (A–Z, a–z), numeric digits (0–9), and the underscore character (_).</p>
brokerID	String	None	<p>Broker identifier</p> <p>For brokers using a JDBC-based persistent data store, this string is appended to the names of all database tables to make them unique in the case where more than one broker instance is using the same database. For brokers using a file-based data store, this property is ignored.</p> <p>In a high-availability cluster, each broker must have a unique broker identifier.</p> <p>The identifier may contain only alphabetic letters (A–Z, a–z), numeric digits (0–9), and the underscore character (_).</p>

ManagedConnectionFactory JavaBean

A *managed connection factory* defines the connections that the Resource Adapter provides to a message-driven bean. [Table 19–2](#) shows the properties of the ManagedConnectionFactory JavaBean; if set, these properties override the corresponding properties of the ResourceAdapter JavaBean.

TABLE 19–2 Managed Connection Factory Properties

Property	Type	Default Value	Description
addressList	String	Inherited from ResourceAdapter JavaBean (see Table 19–1)	List of message service addresses for connecting to Message Queue service
userName ¹	String	guest	User name for connecting to Message Queue service
password ¹	String	guest	Password for connecting to Message Queue service
clientID	String	None	Client identifier for connections to Message Queue service
addressListBehavior	String	PRIORITY	Order in which to attempt connection to Message Queue service: PRIORITY: Order specified in address list RANDOM: Random order Note – Reconnection attempts after a connection failure start with the broker whose connection failed and proceed sequentially through the address list, regardless of the value set for this property.
addressListIterations	Integer	1	Number of times to iterate through address list attempting to establish or reestablish a connection
reconnectEnabled	Boolean	false	Attempt to reestablish a lost connection?
reconnectAttempts	Integer	6	Number of times to attempt reconnection to each address in address list before moving on to next

¹ Optional

TABLE 19-2 Managed Connection Factory Properties (Continued)

Property	Type	Default Value	Description
reconnectInterval	Long integer	30000	Interval, in milliseconds, between reconnection attempts

ActivationSpec JavaBean

Table 19-3 shows the configurable properties of the ActivationSpec JavaBean. These properties are used by the application server when instructing the Resource Adapter to activate a message endpoint and associate it with a message-driven bean.

TABLE 19-3 ActivationSpec Properties

Property	Type	Default Value	Description
addressList ^{1,2}	String	Inherited from ResourceAdapter JavaBean	Message service address for connecting to Message Queue service
destination ³	String	None	Name of destination from which to consume messages The value must be that of the destinationName property for a Message Queue destination administered object.
destinationType ³	String	None	Type of destination specified by destination property: javax.jms.Queue: Queue destination javax.jms.Topic: Topic destination
messageSelector ^{1,3}	String	None	Message selector for filtering messages delivered to consumer
subscriptionName ³	String	None	Name for durable subscriptions This property must be set if subscriptionDurability is set to Durable.

¹ Optional

² Property specific to Message Queue JMS Resource Adapter

³ Standard Enterprise JavaBean (EJB) and J2EE Connector Architecture (CA) property

TABLE 19-3 ActivationSpec Properties (Continued)

Property	Type	Default Value	Description
subscriptionDurability ³	String	NonDurable	<p>Durability of consumer for topic destination:</p> <p>Durable: Durable consumer</p> <p>NonDurable: Nondurable consumer</p> <p>This property is valid only if destinationType is set to javax.jms.Topic, and is optional for nondurable subscriptions and required for durable ones. If set to Durable, the clientId and subscriptionName properties must also be set.</p>
clientId ³	String	None	<p>Client ID for connections to Message Queue service</p> <p>This property must be set if subscriptionDurability is set to Durable.</p>
acknowledgeMode ^{1,3}	String	Auto-acknowledge	<p>Acknowledgment mode:</p> <p>Auto-acknowledge:</p> <p>Auto-acknowledge mode</p> <p>Dups-ok-acknowledge:</p> <p>Dups-OK-acknowledge mode</p>
customAcknowledgeMode	String	None	<p>Acknowledgment mode for MDB message consumption</p> <p>Valid values are No_acknowledge or null.</p> <p>You can use no-acknowledge mode only for a nontransacted, nondurable topic subscription; if you use this setting with a transacted subscription or a durable subscription, subscription activation will fail.</p>
endpointExceptionRedeliveryAttempts	Integer	6	<p>Number of times to redeliver a message when MDB throws an exception during message delivery</p>

³ Standard Enterprise JavaBean (EJB) and J2EE Connector Architecture (CA) property¹ Optional

TABLE 19-3 ActivationSpec Properties (Continued)

Property	Type	Default Value	Description
sendUndeliverableMsgsToDMQ	Boolean	true	<p>Place message in dead message queue when MDB throws a runtime exception and number of redelivery attempts exceeds the value of <code>endpointExceptionRedeliveryAttempts</code>.</p> <p>If false, the Message Queue broker will attempt redelivery of the message to any valid consumer, including the same MDB.</p>

Metrics Information Reference

This chapter describes the metrics information that a Message Queue broker can provide for monitoring, tuning, and diagnostic purposes. This information can be made available in a variety of ways:

- In a log file (see [“Sending Metrics Data to Log Files” on page 217](#))
- Interactively with the Command utility’s `imqcmd metrics` subcommand (see [“Using the Command Utility” on page 84](#))
- In metrics messages sent to a metrics topic destination (see [“Using the Message-Based Monitoring API” on page 224](#))
- Through JMX MBeans that can be accessed programmatically by Java applications using the JMX Administration API.

The tables in this chapter list the kinds of metrics information available and the forms in which it can be provided. For metrics provided through the Command utility’s `imqcmd metrics` subcommand, the tables list the metric type with which they can be requested; for those provided in metrics messages, the tables list the metrics topic destination to which they are delivered. All the metrics information in this chapter can be accessed programmatically using the JMX Administration API as described in the *Message Queue Developer’s Guide for JMX Clients*

The chapter consists of the following sections:

- [“JVM Metrics” on page 356](#)
- [“Brokerwide Metrics” on page 356](#)
- [“Connection Service Metrics” on page 358](#)
- [“Physical Destination Metrics” on page 359](#)

JVM Metrics

Table 20–1 shows the metrics information that the broker reports for the broker process JVM (Java Virtual Machine) heap.

TABLE 20–1 JVM Metrics

Metrics Quantity	Description	Log File?	metrics bkr Metric Type	Metrics Topic
JVM heap: total memory	Current total memory, in bytes	Yes	cxn	mq.metrics.jvm
JVM heap: free memory	Amount of memory currently available for use, in bytes	Yes	cxn	mq.metrics.jvm
JVM heap: max memory	Maximum allowable heap size, in bytes	Yes	None	mq.metrics.jvm

Brokerwide Metrics

Table 20–2 shows the brokerwide metrics information that the broker reports.

TABLE 20–2 Brokerwide Metrics

Metrics Quantity	Description	Log File?	metrics bkr Metric Type	Metrics Topic
Connections				
Num connections	Total current number of connections for all connection services	Yes	cxn	mq.metrics.broker
Num threads	Total current number of threads for all connection services	Yes	cxn	None
Min threads	Total minimum number of threads for all connection services	Yes	cxn	None
Max threads	Total maximum number of threads for all connection services	Yes	cxn	None
Stored Messages				
Num messages	Current number of payload messages stored in memory and persistent store	No	None ¹	mq.metrics.broker
Total message bytes	Total size in bytes of payload messages currently stored in memory and persistent store	No	None ¹	mq.metrics.broker
Message Flow				

¹ Use query bkr command instead

TABLE 20-2 Brokerwide Metrics (Continued)

Metrics Quantity	Description	Log File?	metrics bkr Metric Type	Metrics Topic
Num messages in	Cumulative number of payload messages received since broker started	Yes	ttl	mq.metrics.broker
Num messages out	Cumulative number of payload messages sent since broker started	Yes	ttl	mq.metrics.broker
Rate messages in	Current rate of flow of payload messages into broker	Yes	rts	None
Rate messages out	Current rate of flow of payload messages out of broker	Yes	rts	None
Message bytes in	Cumulative size in bytes of payload messages received since broker started	Yes	ttl	mq.metrics.broker
Message bytes out	Cumulative size in bytes of payload messages sent since broker started	Yes	ttl	mq.metrics.broker
Rate message bytes in	Current rate of flow of payload message bytes into broker	Yes	rts	None
Rate message bytes out	Current rate of flow of payload message bytes out of broker	Yes	rts	None
Num packets in	Cumulative number of payload and control packets received since broker started	Yes	ttl	mq.metrics.broker
Num packets out	Cumulative number of payload and control packets sent since broker started	Yes	ttl	mq.metrics.broker
Rate packets in	Current rate of flow of payload and control packets into broker	Yes	rts	None
Rate packets out	Current rate of flow of payload and control packets out of broker	Yes	rts	None
Packet bytes in	Cumulative size in bytes of payload and control packets received since broker started	Yes	ttl	mq.metrics.broker
Packet bytes out	Cumulative size in bytes of payload and control packets sent since broker started	Yes	ttl	mq.metrics.broker
Rate packet bytes in	Current rate of flow of payload and control packet bytes into broker	Yes	rts	None
Rate packet bytes out	Current rate of flow of payload and control packet bytes out of broker	Yes	rts	None
Destinations				
Num destinations	Current number of physical destinations	No	None	mq.metrics.broker

Connection Service Metrics

Table 20–3 shows the metrics information that the broker reports for individual connection services.

TABLE 20–3 Connection Service Metrics

Metrics Quantity	Description	Log File?	metrics svc Metric Type	Metrics Topic
Connections				
Num connections	Current number of connections	No	cxn ¹	None
Num threads	Current number of threads	No	cxn ¹	None
Min threads	Minimum number of threads assigned to service	No	cxn	None
Max threads	Maximum number of threads assigned to service	No	cxn	None
Message Flow				
Num messages in	Cumulative number of payload messages received through connection service since broker started	No	ttl	None
Num messages out	Cumulative number of payload messages sent through connection service since broker started	No	ttl	None
Rate messages in	Current rate of flow of payload messages into broker through connection service	No	rts	None
Rate messages out	Current rate of flow of payload messages out of broker through connection service	No	rts	None
Message bytes in	Cumulative size in bytes of payload messages received through connection service since broker started	No	ttl	None
Message bytes out	Cumulative size in bytes of payload messages sent through connection service since broker started	No	ttl	None
Rate message bytes in	Current rate of flow of payload message bytes into broker through connection service	No	rts	None
Rate message bytes out	Current rate of flow of payload message bytes out of broker through connection service	No	rts	None
Num packets in	Cumulative number of payload and control packets received through connection service since broker started	No	ttl	None
Num packets out	Cumulative number of payload and control packets sent through connection service since broker started	No	ttl	None

¹ Also available with query svc command

TABLE 20–3 Connection Service Metrics (Continued)

Metrics Quantity	Description	Log File?	metrics svc Metric Type	Metrics Topic
Rate packets in	Current rate of flow of payload and control packets into broker through connection service	No	rts	None
Rate packets out	Current rate of flow of payload and control packets out of broker through connection service	No	rts	None
Packet bytes in	Cumulative size in bytes of payload and control packets received through connection service since broker started	No	ttl	None
Packet bytes out	Cumulative size in bytes of payload and control packets sent through connection service since broker started	No	ttl	None
Rate packet bytes in	Current rate of flow of payload and control packet bytes into broker through connection service	No	rts	None
Rate packet bytes out	Current rate of flow of payload and control packet bytes out of broker through connection service	No	rts	None

Physical Destination Metrics

Table 20–4 shows the metrics information that the broker reports for individual destinations.

TABLE 20–4 Physical Destination Metrics

Metrics Quantity	Description	Log File?	metrics dst Metric Type	Metrics Topic
Message Consumers				
Num consumers	<p>Current number of associated message consumers</p> <p>For queue destinations, this attribute includes both active and backup consumers. For topic destinations, it includes both nondurable and (active and inactive) durable subscribers and is equivalent to “Num active consumers.”</p>	No	con	mq.metrics.destination.queue.queueName mq.metrics.destination.topic.topicName

TABLE 20–4 Physical Destination Metrics (Continued)

Metrics Quantity	Description	Log File?	metrics dst Metric Type	Metrics Topic
Peak num consumers	<p>Peak number of associated message consumers since broker started</p> <p>For queue destinations, this attribute includes both active and backup consumers. For topic destinations, it includes both nondurable and (active and inactive) durable subscribers and is equivalent to “Peak num active consumers.”</p>	No	con	mq.metrics.destination.queue.queueName mq.metrics.destination.topic.topicName
Avg num consumers	<p>Average number of associated message consumers since broker started</p> <p>For queue destinations, this attribute includes both active and backup consumers. For topic destinations, it includes both nondurable and (active and inactive) durable subscribers and is equivalent to “Avg num active consumers.”</p>	No	con	mq.metrics.destination.queue.queueName mq.metrics.destination.topic.topicName
Num active consumers	<p>Current number of associated active message consumers</p> <p>For topic destinations, this attribute includes both nondurable and (active and inactive) durable subscribers and is equivalent to “Num consumers.”</p>	No	con	mq.metrics.destination.queue.queueName mq.metrics.destination.topic.topicName

TABLE 20–4 Physical Destination Metrics (Continued)

Metrics Quantity	Description	Log File?	metrics dst Metric Type	Metrics Topic
Peak num active consumers	Peak number of associated active message consumers since broker started For topic destinations, this attribute includes both nondurable and (active and inactive) durable subscribers and is equivalent to “Peak num consumers.”	No	con	mq.metrics.destination.queue.queueName mq.metrics.destination.topic.topicName
Avg num active consumers	Average number of associated active message consumers since broker started For topic destinations, this attribute includes both nondurable and (active and inactive) durable subscribers and is equivalent to “Avg num consumers.”	No	con	mq.metrics.destination.queue.queueName mq.metrics.destination.topic.topicName
Num backup consumers ¹	Current number of associated backup message consumers	No	con	mq.metrics.destination.queue.queueName mq.metrics.destination.topic.topicName
Peak num backup consumers ¹	Peak number of associated backup message consumers since broker started	No	con	mq.metrics.destination.queue.queueName mq.metrics.destination.topic.topicName
Avg num backup consumers ¹	Average number of associated backup message consumers since broker started	No	con	mq.metrics.destination.queue.queueName mq.metrics.destination.topic.topicName

Stored Messages

Num messages	Current number of messages stored in memory and persistent store	No	con ttl rts ²	mq.metrics.destination.queue.queueName mq.metrics.destination.topic.topicName
--------------	--	----	--------------------------------	--

¹ Queue destinations only² Also available with query dst command

TABLE 20–4 Physical Destination Metrics (Continued)

Metrics Quantity	Description	Log File?	metrics dst Metric Type	Metrics Topic
Num messages remote	Current number of messages stored in memory and persistent store that were sent from a remote broker in a cluster. This number does not include messages included in transactions.	No	Not Available ³	Not Available
Peak num messages	Peak number of messages stored in memory and persistent store since broker started	No	con ttl rts	mq.metrics.destination.queue.queueName mq.metrics.destination.topic.topicName
Avg num messages	Average number of messages stored in memory and persistent store since broker started	No	con ttl rts	mq.metrics.destination.queue.queueName mq.metrics.destination.topic.topicName
Total message bytes	Current total size in bytes of messages stored in memory and persistent store	No	ttl rts ²	mq.metrics.destination.queue.queueName mq.metrics.destination.topic.topicName
Total message bytes remote	Current total size in bytes of messages stored in memory and persistent store that were sent from a remote broker in a cluster. This value does not include messages included in transactions.	No	Not Available ³	Not Available
Peak total message bytes	Peak total size in bytes of messages stored in memory and persistent store since broker started	No	ttl rts	mq.metrics.destination.queue.queueName mq.metrics.destination.topic.topicName

³ Available only with `imqcmd query dst` command² Also available with `query dst` command

TABLE 20–4 Physical Destination Metrics (Continued)

Metrics Quantity	Description	Log File?	metrics dst Metric Type	Metrics Topic
Avg total message bytes	Average total size in bytes of messages stored in memory and persistent store since broker started	No	ttl rts	mq.metrics.destination.queue.queueName mq.metrics.destination.topic.topicName
Message Flow				
Num messages in	Cumulative number of messages received since broker started	No	ttl	mq.metrics.destination.queue.queueName mq.metrics.destination.topic.topicName
Num messages out	Cumulative number of messages sent since broker started	No	ttl	mq.metrics.destination.queue.queueName mq.metrics.destination.topic.topicName
Msg bytes in	Cumulative size in bytes of messages received since broker started	No	ttl	mq.metrics.destination.queue.queueName mq.metrics.destination.topic.topicName
Msg bytes out	Cumulative size in bytes of messages sent since broker started	No	ttl	mq.metrics.destination.queue.queueName mq.metrics.destination.topic.topicName
Peak message bytes	Size in bytes of largest single message received since broker started	No	ttl rts	mq.metrics.destination.queue.queueName mq.metrics.destination.topic.topicName
Rate num messages in	Current rate of flow of messages received	No	rts	None
Rate num messages out	Current rate of flow of messages sent	No	rts	None
Rate msg bytes in	Current rate of flow of message bytes received	No	rts	None
Rate msg bytes out	Current rate of flow of message bytes sent	No	rts	None
Disk Utilization				
Disk reserved ⁴	Amount of disk space, in bytes, reserved for destination	No	dsk	mq.metrics.destination.queue.queueName mq.metrics.destination.topic.topicName
Disk used ⁴	Amount of disk space, in bytes, currently in use by destination	No	dsk	mq.metrics.destination.queue.queueName mq.metrics.destination.topic.topicName

⁴ File-based persistence only

TABLE 20–4 Physical Destination Metrics (Continued)

Metrics Quantity	Description	Log File?	<small>metrics dst</small> Metric Type	Metrics Topic
Disk utilization ratio ⁴	Ratio of disk space in use to disk space reserved for destination	No	dsk	<code>mq.metrics.destination.queue.queueName</code> <code>mq.metrics.destination.topic.topicName</code>

⁴ File-based persistence only

JES Monitoring Framework Reference

This chapter describes the monitoring information items that Message Queue exposes through the Sun Java™ Enterprise System Monitoring Framework (JESMF), using the Monitoring Framework's Common Monitoring Model (CMM). It contains the following sections:

- “Common Attributes” on page 365
- “Message Queue Product Information” on page 366
- “Broker Information” on page 366
- “Port Mapper Information” on page 367
- “Connection Service Information” on page 367
- “Destination Information” on page 369
- “Persistent Store Information” on page 370
- “User Repository Information” on page 370

Common Attributes

The attributes listed in [Table 21–1](#) are common to all (or almost all) CMM objects.

TABLE 21–1 JESMF Common Object Attributes

Attribute	Description
Name	Object name
Caption	Short description
Description	Full description
LastUpdateTime	Time last updated
OperationalStatus	Current status (for example, OK or DORMANT)
StatusDescriptions	Description of status

TABLE 21–1 JESMF Common Object Attributes (Continued)

Attribute	Description
OperationalStatusLastChange	Time of last change in operational status

Message Queue Product Information

Table 21–2 shows attributes of the Message Queue product itself that can be accessed with JESMF.

TABLE 21–2 JESMF-Accessible Message Queue Product Attributes

Attribute	Description
ProductName	Product name
ProductIdentifyingNumber	Identifying number of product, in the form <i>urn:uuid:xxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx</i> Value changes for every version.
ProductVendor	Vendor name
ProductVersion	Version number
RevisionNumber	Revision number
BuildNumber	Build number
PatchID	Patch identifier (if any)
CollectionID	Identification key for installed product object Differentiates among product installations; usually identifies the installation location.
InstallDate	Installation date

Broker Information

Table 21–3 shows the JESMF-accessible attributes pertaining to each broker instance.

TABLE 21–3 JESMF-Accessible Message Queue Broker Attributes

Attribute	Description
PrimaryOwnerName	Name of primary system owner (broker property <code>imq.primaryowner.name</code> ; see Table 16–10)

TABLE 21–3 JESMF-Accessible Message Queue Broker Attributes (Continued)

Attribute	Description
PrimaryOwnerContact	Contact information for primary system owner (broker property <code>imq.primaryowner.contact</code> ; see Table 16–10)
Roles	Array of strings denoting broker's roles (taken from broker properties <code>imq.broker.adminDefinedRoles.names</code> ; see Table 16–10)
StartupTime	Time of last startup (date and time in milliseconds)
URL	URL of Port Mapper
ConfigurationDirectory	Broker instance directory (for example, <code>/var/imq/instances/mybroker</code>)
DirectoryName	Distinguished name of directory (for example, LDAP) entry where static information about application is stored An empty string indicates that no information about the application is available in the directory.

Port Mapper Information

The attributes shown in [Table 21–4](#) provide information about a broker's Port Mapper.

TABLE 21–4 JESMF-Accessible Message Queue Port Mapper Attributes

Attribute	Description
LabeledURI	URI for accessing Port Mapper, in the form <code>mq://hostName:portNumber</code>
Secured	Is Port Mapper access secure (SSL/TLS)?

Connection Service Information

[Table 21–5](#) shows the JESMF-accessible attributes pertaining to each connection service.

TABLE 21-5 JESMF-Accessible Message Queue Connection Service Attributes

Attribute	Description
LabeledURI	URI for accessing connection service, in the form <code>mq://hostName:portNumber/serviceName</code> if dynamically allocated, or <code>mqtcp://hostName:servicePort/serviceName</code> or <code>mqssl://hostName:servicePort/serviceName</code> if statically assigned
Secured	Is connection service access secure (SSL/TLS)?
ConnectionsCount	Current number of connections
NumConnectionsCreated	Cumulative number of connections created since broker started
FailedConnectionsCount	Cumulative number of connections rejected since broker started
CurrentNumberOfThreads	Current number of threads actively handling connections
MinThreadPoolSize	Minimum number of threads maintained in connection service's thread pool (broker property <code>imq.serviceName.min_threads</code> ; see Table 16-1)
MaxThreadPoolSize	Number of threads beyond which no new threads are added to thread pool for use by connection service (broker property <code>imq.serviceName.max_threads</code> ; see Table 16-1)
NumProducers	Current number of message producers
NumConsumers	Current number of message consumers
NumMsgsIn	Cumulative number of messages received since broker started
NumMsgsOut	Cumulative number of messages sent since broker started
InBytesCount	Cumulative size in bytes of messages received since broker started
OutBytesCount	Cumulative size in bytes of messages sent since broker started
NumPktsIn	Cumulative number of packets received since broker started
NumPktsOut	Cumulative number of packets sent since broker started
PktBytesIn	Cumulative size in bytes of packets received since broker started
PktBytesOut	Cumulative size in bytes of packets sent since broker started

Destination Information

Table 21–6 shows the JESMF-accessible attributes pertaining to each destination. Each of these attributes corresponds to a Message Queue physical destination property; see Table 17–1 for further information.

TABLE 21–6 JESMF-Accessible Message Queue Destination Attributes

Attribute	Corresponding Property	Description
Type		Destination type (q = queue, t = topic)
MaxNumMsgs	maxNumMsgs	Maximum number of unconsumed messages
MaxBytesPerMsg	maxBytesPerMsg	Maximum size, in bytes, of any single message
MaxTotalMsgBytes	maxTotalMsgBytes	Maximum total memory, in bytes, for unconsumed messages
LimitBehavior	limitBehavior	Broker behavior when memory-limit threshold reached
MaxNumProducers ¹	maxNumProducers	Maximum number of associated message producers
MaxNumActiveConsumers ²	maxNumActiveConsumers	Maximum number of associated active message consumers in load-balanced delivery
MaxNumBackupConsumers ²	maxNumBackupConsumers	Maximum number of associated backup message consumers in load-balanced delivery
ConsumerFlowLimit	consumerFlowLimit	Maximum number of messages delivered to consumer in a single batch
LocalOnly ¹	isLocalOnly	Local delivery only?
LocalDeliveryPreferred ^{1,2}	localDeliveryPreferred	Local delivery preferred?
UseDMQ ¹	useDMQ	Send dead messages to dead message queue?

¹ Does not apply to dead message queue

² Queue destinations only

Persistent Store Information

The attributes shown in [Table 21–7](#) pertain to the persistent data store.

TABLE 21–7 JESMF-Accessible Message Queue Persistent Store Attributes

Attribute	Description
AccessInfo	URL for accessing JDBC database
InfoFormat	Format of AccessInfo attribute (URL)
JDBCDriver	JDBC driver
UserName	User name for authentication

User Repository Information

The attributes shown in [Table 21–8](#) pertain to the LDAP user repository.

TABLE 21–8 JESMF-Accessible Message Queue User Repository Attributes

Attribute	Description
AccessInfo	URL for accessing LDAP server
InfoFormat	Format of AccessInfo attribute (URL)
Base	Root or base node for user lookup
GroupBase	Root or base node for group lookup
UserName	User name for authentication

PART IV

Appendixes

- [Appendix A, “Platform-Specific Locations of Message Queue Data”](#)
- [Appendix B, “Stability of Message Queue Interfaces”](#)
- [Appendix C, “HTTP/HTTPS Support”](#)
- [Appendix D, “JMX Support”](#)
- [Appendix E, “Frequently Used Command Utility Commands”](#)

Platform-Specific Locations of Message Queue Data

Sun Java™ System Message Queue data is stored in different locations on different operating system platforms. The tables that follow show the location of various types of Message Queue data on the following platforms:

- “Solaris OS” on page 373
- “Linux” on page 375
- “Windows” on page 376

In the tables, *instanceName* denotes the name of the broker instance with which the data is associated.

Solaris OS

Table A-1 shows the location of Message Queue data on the Solaris operating system. If you are using Message Queue on Solaris with the standalone version of Sun Java System Application Server, the directory structure is like that described under “Windows” on page 376.

TABLE A-1 Message Queue Data Locations on Solaris Platform

Data Category	Location
Command line executable files	/usr/bin
Broker instance configuration properties	/var/imq/instances/ <i>instanceName</i> /props/config.properties
Broker configuration file templates	/usr/share/lib/imq/props/broker/

TABLE A-1 Message Queue Data Locations on Solaris Platform (Continued)

Data Category	Location
Persistent data store (messages, destinations, durable subscriptions, transactions, acknowledgements)	<code>/var/imq/instances/instanceName/fs370</code> or a JDBC-accessible data store
Broker instance log file directory (default location)	<code>/var/imq/instances/instanceName/log/</code>
Administered objects (object store)	Local directory of your choice or an LDAP server
Security: user repository	<code>/var/imq/instances/instanceName/etc/passwd</code> or an LDAP server
Security: access control file (default location)	<code>/var/imq/instances/instanceName/etc/accesscontrol.properties</code>
Security: password file directory (default location)	<code>/var/imq/instances/instanceName/etc/</code>
Security: example password file	<code>/etc/imq/passfile.sample</code>
Security: broker's key store file location	<code>/etc/imq/</code>
JavaDoc API documentation	<code>/usr/share/javadoc/imq/index.html</code>
Example applications and configurations	<code>/usr/demo/imq/</code>
Java archive (.jar), Web archive (.war), and Resource Adapter archive (.rar) files	<code>/usr/share/lib/imq</code>
External resource (.jar) files such as JDBC drivers, JAAS login modules, and so forth	<code>/usr/share/lib/imq/ext</code>

Linux

Table A–2 shows the location of Message Queue data on the Linux operating system.

TABLE A–2 Message Queue Data Locations on Linux Platform

Data Category	Location
Command line executable files	/opt/sun/mq/bin
Broker instance configuration properties	/var/opt/sun/mq/instances/ <i>instanceName</i> /props/config.properties
Broker configuration file templates	/opt/sun/mq/private/share/lib/props/
Persistent data store (messages, destinations, durable subscriptions, transactions, acknowledgements)	/var/opt/sun/mq/instances/ <i>instanceName</i> /fs370/ or a JDBC-accessible data store
Broker instance log file directory (default location)	/var/opt/sun/mq/instances/ <i>instanceName</i> /log/
Administered objects (object store)	Local directory of your choice or an LDAP server
Security: user repository	/var/opt/sun/mq/instances/ <i>instanceName</i> /etc/passwd or an LDAP server
Security: access control file (default location)	/var/opt/sun/mq/instances/ <i>instanceName</i> /etc/accesscontrol.properties
Security: password file directory (default location)	/var/opt/sun/mq/instances/ <i>instanceName</i> /etc/
Security: example password file	/etc/opt/sun/mq/passfile.sample
Security: broker's key store file location	/etc/opt/sun/mq/
JavaDoc API documentation	/opt/sun/mq/javadoc/index.html
Example applications and configurations	/opt/sun/mq/examples/

TABLE A-2 Message Queue Data Locations on Linux Platform (Continued)

Data Category	Location
Java archive (.jar), Web archive (.war), and Resource Adapter archive (.rar) files	/opt/sun/mq/share/lib/
External resource (.jar) files such as JDBC drivers, JAAS login modules, and so forth	/opt/sun/mq/share/lib/ext
Shared library (.so) files	/opt/sun/mq/lib/

Windows

Table A-3 shows the location of Message Queue data on the Windows operating system. The table also applies to the Solaris platform when Message Queue is bundled with the standalone version of Sun Java System Application Server. That version of Application Server is bundled with neither Solaris nor Sun Java Enterprise System. Use the pathnames in Table A-3, but change the direction of the slash characters from the Windows backslash (\) to the Solaris forward slash (/). See “[Directory Variable Conventions](#)” on page 25 for definitions of the IMQ_HOME and IMQ_VARHOME directory variables.

TABLE A-3 Message Queue Data Locations on Windows Platform

Data Category	Location
Command line executable files	IMQ_HOME\bin
Broker instance configuration properties	IMQ_VARHOME\instances\instanceName\props\config.properties
Broker configuration file templates	IMQ_HOME\lib\props\broker\
Persistent data store (messages, destinations, durable subscriptions, transactions, acknowledgements)	IMQ_VARHOME\instances\instanceName\fs370\ or a JDBC-accessible data store
Broker instance log file directory (default location)	IMQ_VARHOME\instances\instanceName\log\
Administered objects (object store)	Local directory of your choice or an LDAP server

TABLE A-3 Message Queue Data Locations on Windows Platform (Continued)

Data Category	Location
Security: user repository	IMQ_VARHOME\instances\ <i>instanceName</i> \etc\passwd or an LDAP server
Security: access control file (default location)	IMQ_VARHOME\instances\ <i>instanceName</i> \etc\accesscontrol.properties
Security: password file directory (default location)	IMQ_HOME\etc\
Security: example password file	IMQ_HOME\etc\passfile.sample
Security: broker's key store file location	IMQ_HOME\etc\
JavaDoc API documentation	IMQ_HOME\javadoc\index.html
Example applications and configurations	IMQ_HOME\demo\
Java archive (.jar), Web archive (.war), and Resource Adapter archive (.rar) files	IMQ_HOME\lib\
External resource (.jar) files such as JDBC drivers, JAAS login modules, and so forth	IMQ_HOME\lib\ext

Stability of Message Queue Interfaces

Sun Java™ System Message Queue uses many interfaces that can help administrators automate tasks. This appendix classifies the interfaces according to their stability. The more stable an interface is, the less likely it is to change in subsequent versions of the product.

Any interface that is not listed in this appendix is private and not for customer use.

Classification Scheme

Appendix B, “Stability of Message Queue Interfaces,” describes the stability classification scheme.

TABLE B-1 Interface Stability Classification Scheme

Classification	Description
Private	Not for direct use by customers. May change or be removed in any release.
Evolving	For use by customers. Subject to incompatible change at a major (e.g. 3.0, 4.0) or minor (e.g. 3.1, 3.2) release. The changes will be made carefully and slowly. Reasonable efforts will be made to ensure that all changes are compatible but that is not guaranteed.
Stable	For use by customers. Subject to incompatible change at a major (for example, 3.0 or 4.0) release only.
Standard	For use by customers. These interfaces are defined by a formal standard, and controlled by a standards organization. Incompatible changes to these interfaces are rare.

TABLE B-1 Interface Stability Classification Scheme (Continued)

Classification	Description
Unstable	For use by customers. Subject to incompatible change at a major (e.g. 3.0, 4.0) or minor (e.g. 3.1, 3.2) release. Customers are advised that these interfaces may be removed or changed substantially and in an incompatible way in a future release. It is recommended that customers not create explicit dependencies on unstable interfaces.

Interface Stability

Appendix B, “Stability of Message Queue Interfaces,” lists the interfaces and their classifications.

TABLE B-2 Stability of Message Queue Interfaces

Interface	Classification
Command Line Interfaces	
imqbrokerd command line interface	Evolving
imqadmin command line interface	Unstable
imqcmd command line interface	Evolving
imqdbmgr command line interface	Unstable
imqkeytool command line interface	Evolving
imqobjmgr command line interface	Evolving
imqusermgr command line interface	Unstable
Output from imqbrokerd, imqadmin, imqcmd, imqdbmgr, imqkeytool, imqobjmgr, imqusermgr	Unstable
Commands	
imqobjmgr command file	Evolving
imqbrokerd command	Stable
imqadmin command	Unstable
imqcmd command	Stable
imqdbmgr command	Unstable
imqkeytool command	Stable
imqobjmgr command	Stable

TABLE B-2 Stability of Message Queue Interfaces (Continued)

Interface	Classification
imqusermgr command	Unstable
APIs	
JMS API (javax.jms)	Standard
JAXM API (javax.xml)	Standard
C-API	Evolving
C-API environment variables	Unstable
Message-based monitoring API	Evolving
Administered Object API (com.sun.messaging)	Evolving
.jar and .war Files	
imq.jar location and name	Stable
jms.jar location and name	Evolving
imqbroker.jar location and name	Private
imqutil.jar location and name	Private
imqadmin.jar location and name	Private
imqservlet.jar location and name	Evolving
imqhttp.war location and name	Evolving
imqhttps.war location and name	Evolving
imqjmsra.rar location and name	Evolving
imqxm.jar location and name	Evolving
jaxm-api.jar location and name	Evolving
saa-j-api.jar location and name	Evolving
saa-j-impl.jar location and name	Evolving
activation.jar location and name	Evolving
mail.jar location and name	Evolving
dom4j.jar location and name	Private
fscontext.jar location and name	Unstable
Files	
Broker log file location and content format	Unstable

TABLE B-2 Stability of Message Queue Interfaces (Continued)

Interface	Classification
password file	Unstable
accesscontrol.properties file	Unstable
System Destinations	
mq.sys.dmq destination	Stable
mq.metrics.* destinations	Evolving
Configuration Properties	
Message Queue JMS Resource Adapter configuration properties	Evolving
Message Queue JMS Resource Adapter JavaBean and ActivationSpec configuration properties	Evolving
Message Properties and Formats	
Dead message queue message property, JMSXDeliveryCount	Standard
Dead message queue message properties, JMS_SUN_*	Evolving
Message Queue client message properties: JMS_SUN_*	Evolving
JMS message format for metrics or monitoring messages	Evolving
Miscellaneous	
Message Queue JMS Resource Adapter package, com.sun.messaging.jms.ra	Evolving
JDBC schema for storage of persistent messages	Evolving

HTTP/HTTPS Support

Message Queue includes support for Java clients to communicate with a message broker by means of the HTTP or secure HTTP (HTTPS) transport protocols, rather than through a direct TCP connection. (HTTP/HTTPS support is not available for C clients.) Because HTTP/HTTPS connections are normally allowed through firewalls, this allows client applications to be separated from the broker by a firewall.

This appendix describes the architecture used to enable HTTP/HTTPS support and explains the setup work needed to allow Message Queue clients to use such connections. It has the following sections:

- [“HTTP/HTTPS Support Architecture” on page 383](#)
- [“Enabling HTTP/HTTPS Support” on page 384](#)
- [“Troubleshooting” on page 398](#)

HTTP/HTTPS Support Architecture

Message Queue’s support architecture is very similar for both HTTP and HTTPS support, as shown in [Figure C-1](#):

- On the client side, an HTTP or HTTPS transport driver (part of the Message Queue client runtime) encapsulates each message into an HTTP request and makes sure that these requests are transmitted in the correct sequence.
- If necessary, the client can use an HTTP proxy server to communicate with the broker. The proxy’s address is specified using command line options when starting the client; see [“Using an HTTP Proxy” on page 398](#) for more information.
- An HTTP or HTTPS tunnel servlet (both bundled with Message Queue) is loaded into an application server or Web server on the broker side and used to pull payload messages from client HTTP requests before forwarding them to the broker. The tunnel servlet also sends broker messages back to the client in response to the client’s HTTP requests. A single tunnel servlet can be used to access multiple brokers.

- On the broker side, the `httpjms` or `httpsjms` connection service unwraps and demultiplexes incoming messages from the corresponding tunnel servlet.

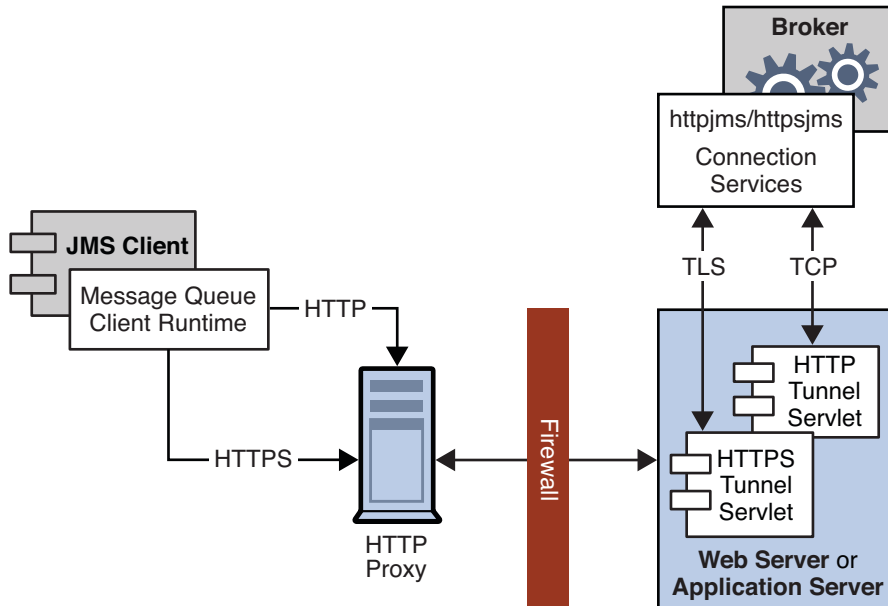


FIGURE C-1 HTTP/HTTPS Support Architecture

The main difference between HTTP and HTTPS connections is that in the HTTPS case (`httpsjms` connection service), the tunnel servlet has a secure connection to both the client application and the broker. The secure connection to the broker is established by means of the Secure Socket Layer (SSL) protocol. Message Queue's SSL-enabled HTTPS tunnel servlet passes a self-signed certificate to any broker requesting a connection. The broker uses the certificate to establish an encrypted connection to the tunnel servlet. Once this connection is established, a secure connection between the client application and the tunnel servlet can be negotiated by the client application and the application server or Web server.

Enabling HTTP/HTTPS Support

The procedures for enabling HTTP and HTTPS support are essentially the same for both protocols, although a few extra steps are required in the HTTPS case to generate and access the needed encryption keys and certificates. The steps are as follows. (For HTTPS, start with step 1; for non-secure HTTP, start with step 4.)

1. (*HTTPS only*) Generate a self-signed certificate for the HTTPS tunnel servlet.

2. (*HTTPS only*) Modify the deployment descriptor in the tunnel servlet's `.war` file to specify the location and password of the certificate key store.
3. (*HTTPS only*) Validate the Web or application server's self-signed certificate and install it in the client application's trust store.
4. (*HTTP and HTTPS*) Deploy the HTTP or HTTPS tunnel servlet.
5. (*HTTP and HTTPS*) Configure the broker's `httpjms` or `httpsjms` connection service and start the broker.
6. (*HTTP and HTTPS*) Configure an HTTP or HTTPS connection.

The following subsections describe each of these steps in greater detail, using Sun Java™ System Application Server as an example for purposes of illustration. If you are using a different application server or Web server (such as Sun Java System Web Server), the procedures will be substantially similar but may differ in detail; see your server product's own documentation for specifics.

Step 1 (HTTPS Only): Generating a Self-Signed Certificate for the Tunnel Servlet

Message Queue's SSL support is oriented toward securing on-the-wire data, on the assumption that the client is communicating with a known and trusted server. Therefore, SSL is implemented using only self-signed server certificates. Before establishing an HTTPS connection, you must obtain such a certificate. (This step is not needed for ordinary, non-secure HTTP connections.)

Run the Message Queue Key Tool utility (`imqkeytool`) to generate a self-signed certificate for the tunnel servlet. (On UNIX systems, you may need to run the utility as the root user in order to have permission to create the key store.) Enter the following at the command prompt:

```
imqkeytool -servlet keyStoreLocation
```

where *keyStoreLocation* is the location of Message Queue's key store file.

The Key Tool utility prompts you for a key store password:

```
Enter keystore password:
```

After you have entered a valid password, the utility prompts you for identifying information from which to construct an X.500 distinguished name. [Table C-1](#) shows the prompts and the values to be provided for each prompt. Values are case-insensitive and can include spaces.

TABLE C-1 Distinguished Name Information Required for a Self-Signed Certificate

Prompt	X.500 Attribute	Description	Example
What is your first and last name?	commonName (CN)	Fully qualified name of server running the broker	mqserver.sun.com
What is the name of your organizational unit?	organizationalUnit (OU)	Name of department or division	purchasing
What is the name of your organization?	organizationName (ON)	Name of larger organization, such as a company or government entity	Acme Widgets, Inc.
What is the name of your city or locality?	localityName (L)	Name of city or locality	San Francisco
What is the name of your state or province?	stateName (ST)	Full (unabbreviated) name of state or province	California
What is the two-letter country code for this unit?	country (C)	Standard two-letter country code	US

When you have entered the information, the Key Tool utility displays it for confirmation: for example,

```
Is CN=mqserver.sun.com, OU=purchasing, ON=Acme Widgets, Inc.,
L=San Francisco, ST=California, C=US correct?
```

To accept the current values and proceed, enter yes; to reenter values, accept the default or enter no. After you confirm, the utility pauses while it generates a key pair.

Next, the utility asks for a password to lock the key pair (key password). Press Return in response to this prompt to use the same password for both the key password and the key store password.



Caution – Be sure to remember the password you specify. You must provide this password later to the tunnel servlet so it can open the key store.

The Key Tool utility generates a self-signed certificate and places it in Message Queue’s key store file at the location you specified for the *keyStoreLocation* argument.



Caution – The HTTPS tunnel servlet must be able to see the key store. Be sure to move or copy the generated key store from the location specified by *keyStoreLocation* to one accessible to the tunnel servlet (see “[Step 4 \(HTTP and HTTPS\): Deploying the Tunnel Servlet](#)” on page 392).

Step 2 (HTTPS Only): Specifying the Key Store Location and Password

The tunnel servlet's Web archive (.war) file includes a *deployment descriptor*, an XML file containing the basic configuration information needed by the application server or Web server to load and run the servlet. Before deploying the .war file for the HTTPS tunnel servlet, you must edit the deployment descriptor to specify the location and password of the certificate key store. (This step is not needed for ordinary, non-secure HTTP connections.)

▼ To Specify the Location and Password of the Certificate Key Store

1 Copy the .war file to a temporary directory:

The location of the HTTPS tunnel servlet's .war file varies, depending on your platform (see [Appendix A, "Platform-Specific Locations of Message Queue Data"](#)):

Solaris: `cp /usr/share/lib/imq/imqhttps.war /tmp`

Linux: `cp /opt/sun/mq/share/lib/imqhttps.war /tmp`

Windows: `cp IMQ_HOME\lib\imqhttps.war \tmp`

2 Make the temporary directory your current directory.

```
cd /tmp
```

3 Extract the contents of the .war file.

```
jar xvf imqhttps.war
```

4 List the .war file's deployment descriptor.

Enter the command

```
ls -l WEB-INF/web.xml
```

to confirm that the deployment descriptor file (WEB-INF/web.xml) was successfully extracted.

5 Edit the deployment descriptor to specify the key store location and password.

Edit the web.xml file to provide appropriate values for the `keyStoreLocation` and `keyStorePassword` elements (as well as `servletPort` and `servletHost`, if necessary): for example,

```
<init-param>
  <param-name>keyStoreLocation</param-name>
  <param-value>/local/tmp/imqhttps/keystore</param-value>
</init-param>
<init-param>
  <param-name>keyStorePassword</param-name>
  <param-value>shazam</param-value>
```

```
</init-param>
<init-param>
  <param-name>servletHost</param-name>
  <param-value>localhost</param-value>
</init-param>
<init-param>
  <param-name>servletPort</param-name>
  <param-value>7674</param-value>
</init-param>
```

Note – If you are concerned about exposure of the key store password, you can use file-system permissions to restrict access to the `imqhttps.war` file.)

6 Reassemble the contents of the .war file.

```
jar uvf imqhttps.war WEB-INF/web.xml
```

Step 3 (HTTPS Only): Validating and Installing the Server's Self-Signed Certificate

In order for a client application to communicate with the Web or application server, you must validate the server's self-signed certificate and install it in the application's trust store. The following procedure shows how:

▼ To Validate and Install the Server's Self-Signed Certificate

1 Validate the server's certificate.

By default, the Sun Java System Application Server generates a self-signed certificate and stores it in a key store file at the location

appServerRoot/glassfish/domains/domain1/config/keystore.jks

where *appServerRoot* is the root directory in which the Application Server is installed.

Note – If necessary, you can use the JDK Key Tool utility to generate a key store of your own and use it in place of the default key store. For more information, see the section “Establishing a Secure Connection Using SSL” in Chapter 28, “Introduction to Security in Java EE,” of the *Java EE 5 Tutorial* at

<http://java.sun.com/javaee/5/docs/tutorial/doc/Security-Intro7.html>

a. Make the directory containing the key store file your current directory.

For example, to use the Application Server's default key store file (as shown above), navigate to its directory with the command

```
cd appServerRoot/glassfish/domains/domain1/config
```

where *appServerRoot* is, again, the root directory in which the Application Server is installed.

b. List the contents of the key store file.

The Key Tool utility's `-list` option lists the contents of a specified key store file. For example, the following command lists the Application Server's default key store file (`keystore.jks`):

```
keytool -list -keystore keystore.jks -v
```

The `-v` option tells the Key Tool utility to display certificate fingerprints in human-readable form.

c. Enter the key store password.

The Key Tool utility prompts you for the key store file's password:

```
Enter keystore password:
```

By default, the key store password is set to `changeit`; you can use the Key Tool utility's `-storepasswd` option to change it to something more secure. After you have entered a valid password, the Key Tool utility will respond with output like the following:

```
Keystore type: JKS
Keystore provider: SUN

Your keystore contains 1 entry

Alias name: slas
Creation date: Nov 13, 2007
Entry type: PrivateKeyEntry
Certificate chain length: 1
Certificate[1]:
Owner: CN=helios, OU=Sun Java System Application Server, O=Sun Microsystems,
L=Santa Clara, ST=California, C=US
Issuer: CN=helios, OU=Sun Java System Application Server, O=Sun Microsystems,
L=Santa Clara, ST=California, C=US
Serial number: 45f74784
Valid from: Tue Nov 13 13:18:39 PST 2007 until: Fri Nov 10 13:18:39 PST 2017
Certificate fingerprints:
    MD5:  67:04:CC:39:83:37:2F:D4:11:1E:81:20:05:98:0E:D9
    SHA1: A5:DE:D8:03:96:69:C5:55:DD:E1:C4:13:C1:3D:1D:D0:4C:81:7E:CB
Signature algorithm name: MD5withRSA
Version: 1
```

d. Verify the certificate's fingerprints.

Obtain the correct fingerprints for the Application Server's self-signed certificate by independent means (such as by telephone) and compare them with the fingerprints displayed by the `keytool -list` command. *Do not* accept the certificate and install it in your application's trust store unless the fingerprints match.

2 Export the Application Server's certificate to a certificate file.

Use the Key Tool utility's `-export` option to export the certificate from the Application Server's key store to a separate certificate file, from which you can then import it into your application's trust store. For example, the following command exports the certificate shown above, whose alias is `slas`, from the Application Server's default key store (`keystore.jks`) to a certificate file named `slas.cer`:

```
keytool -export -keystore keystore.jks -storepass changeit
        -alias slas -file slas.cer
```

The Key Tool utility responds with the output

```
Certificate stored in file <slas.cer>
```

3 Verify the contents of the certificate file.

If you wish, you can double-check the contents of the certificate file to make sure it contains the correct certificate:

a. List the contents of the certificate file.

The Key Tool utility's `-printcert` option lists the contents of a specified certificate file. For example, the following command lists the certificate file `slas.cer` that was created in the preceding step:

```
keytool -printcert -file slas.cer -v
```

Once again, the `-v` option tells the Key Tool utility to display the certificate's fingerprints in human-readable form. The resulting output looks like the following:

```
Owner: CN=helios, OU=Sun Java System Application Server, O=Sun Microsystems,
L=Santa Clara, ST=California, C=US
Issuer: CN=helios, OU=Sun Java System Application Server, O=Sun Microsystems,
L=Santa Clara, ST=California, C=US
Serial number: 45f74784
Valid from: Tue Nov 13 13:18:39 PST 2007 until: Fri Nov 10 13:18:39 PST 2017
Certificate fingerprints:
    MD5:  67:04:CC:39:83:37:2F:D4:11:1E:81:20:05:98:0E:D9
    SHA1: A5:DE:D8:03:96:69:C5:55:DD:E1:C4:13:C1:3D:1D:D0:4C:81:7E:CB
    Signature algorithm name: MD5withRSA
    Version: 1
```

b. Confirm the certificate's contents.

Examine the output from the `keytool -printcert` command to make sure that the certificate is correct.

4 Import the certificate into your application's trust store.

The Key Tool utility's `-import` option installs a certificate from a certificate file in a specified trust store. For example, if your client application's trust store is kept in the file `/local/tmp/imqhttps/appKeyStore`, the following command will install the certificate from the file `slas.cer` created above:

```
keytool -import -file slas.cer -keystore "/local/tmp/imqhttps/appKeyStore"
```

Step 4 (HTTP and HTTPS): Deploying the Tunnel Servlet

You can deploy the HTTP or HTTPS tunnel servlet on Sun Java System Application Server either from the command line or by using the Application Server's Web-based administration GUI. In either case, you must then modify the Application Server's security policy file to grant permissions for the tunnel servlet.

To deploy the tunnel servlet from the command line, use the `deploy` subcommand of the Application Server administration utility (`asadmin`): for example,

```
asadmin deploy --user admin --passwordfile pfile.txt --force=true
/local/tmp/imqhttps/imqhttps.war
```

The procedure below shows how to use the Web-based GUI to deploy the servlet.

After deploying the tunnel servlet (whether from the command line or with the Web-based GUI), proceed to [“Modifying the Application Server's Security Policy File” on page 393](#) for instructions on how to grant it the appropriate permissions.

▼ To Deploy the HTTP or HTTPS Tunnel Servlet

1 Deploy the tunnel servlet:

a. In the Web-based administration GUI, choose

App Server>Instances>*appServerInstance*>Applications>Web Applications

where *appServerInstance* is the Application Server instance on which you are deploying the tunnel servlet.

b. Click the Deploy button.

2 Specify the .war file location:

a. Enter the location of the tunnel servlet's Web archive file (*imqhttp.war* or *imqhttps.war*) in the File Path text field.

The file is located in the Message Queue installation directory containing `.jar`, `.war`, and `.rar` files, depending on your operating system platform (see [Appendix A, “Platform-Specific Locations of Message Queue Data”](#)).

b. Click the OK button.

3 Specify the context root directory:

a. Enter the */contextRoot* portion of the tunnel servlet's URL.

The URL has the form

```
http://hostName:portNumber/contextRoot/tunnel
```

or

```
https://hostName:portNumber/contextRoot/tunnel
```

For example, if the URL for the tunnel servlet is

```
http://hostName:portNumber/imq/tunnel
```

the value you enter would be

```
/imq
```

b. Click the OK button.

A confirmation screen appears, showing that the tunnel servlet has been successfully deployed and is enabled by default. The servlet is now available at the URL

```
http://hostName:portNumber/contextRoot/tunnel
```

or

```
https://hostName:portNumber/contextRoot/tunnel
```

where *contextRoot* is the context root directory you specified in step a above. Clients can now use this URL to connect to the message service using an HTTP or HTTPS connection.

4 Modify the server's security policy file

Once you have deployed the HTTP or HTTPS tunnel servlet, you must grant it the appropriate permissions by modifying the Application Server's security policy file, as described in the next procedure.

▼ Modifying the Application Server's Security Policy File

Each Application Server instance has a *security policy file* specifying its security policies or rules. Unless modified, the default security policies would prevent the HTTP or HTTPS tunnel servlet from accepting connections from the Message Queue message broker. In order for the broker to connect to the tunnel servlet, you must add an additional entry to this policy file:

1 Open the security policy file.

The file is named `server.policy` and resides at a location that varies depending on your operating system platform. On the Solaris platform, for example, the policy file for server jeeves would be located at

```
appServerRoot/glassfish/domains/domain1/jeeves/config/server.policy
```

where *appServerRoot* is the root directory in which Sun Java System Application Server is installed.

2 Add the following entry to the file:

```
grant codeBase
    "file:appServerRoot/glassfish/domains/domain1/jeeves
        /applications/j2ee-modules/imqhttps/-
    {
        permission java.net.SocketPermission "*", "connect,accept,resolve";
    };
```

3 Save and close the security policy file.

Step 5 (HTTP and HTTPS): Configuring the Connection Service

HTTP/HTTPS support is not activated for a broker by default, so before connecting using these protocols, you need to reconfigure the broker to activate the `httpjms` or `httpsjms` connection service. [Table C–2](#) shows broker configuration properties pertaining specifically to these two connection services. Once reconfigured, the broker can be started normally, as described under “[Starting Brokers](#)” on page 68.

TABLE C–2 Broker Configuration Properties for the `httpjms` and `httpsjms` Connection Services

Property	Type	Default Value	Description
<code>imq.httpjms.http.servletHost</code> <code>imq.httpsjms.https.servletHost</code>	String	<code>localhost</code>	Host name or IP address of (local or remote) host running tunnel servlet
<code>imq.httpjms.http.servletPort</code> <code>imq.httpsjms.https.servletPort</code>	Integer	<code>httpjms: 7675</code> <code>httpsjms: 7674</code>	Port number of tunnel servlet
<code>imq.httpjms.http.pullPeriod</code> <code>imq.httpsjms.https.pullPeriod</code>	Integer	<code>–1</code>	Interval, in seconds, between client HTTP/HTTPS requests If zero or negative, the client will keep one request pending at all times.
<code>imq.httpjms.http.connectionTimeout</code> <code>imq.httpsjms.https.connectionTimeout</code>	Integer	<code>60</code>	Tunnel servlet timeout interval

▼ To Activate the `httpjms` or `httpsjms` Connection Service

1 Open the broker’s instance configuration file.

The instance configuration file is named `config.properties` and is located in a directory identified by the name of the broker instance to which it belongs:

```
.../instances/instanceName/props/config.properties
```

(See [Appendix A, “Platform-Specific Locations of Message Queue Data,”](#) for the location of the instances directory.)

2 Add httpjms or httpsjms to the list of active connection services.

Add the value httpjms or httpsjms to the `imq.service.activelist` property: for example,

```
imq.service.activelist=jms,admin,httpjms
```

or

```
imq.service.activelist=jms,admin,httpsjms
```

3 Set any other HTTP/HTTPS-related configuration properties as needed.

At startup, the broker looks for an application server or Web server and an HTTP or HTTPS tunnel servlet running on its local host machine. If necessary, you can reconfigure the broker to access a remote tunnel servlet instead, by setting the `servletHost` and `servletPort` properties appropriately (see [Table C-2](#)): for example,

```
imq.httpjms.http.servletHost=helios
imq.httpjms.http.servletPort=7675
```

You can also improve performance by reconfiguring the connection service’s `pullPeriod` property. This specifies the interval, in seconds, at which each client issues HTTP/HTTPS requests to pull messages from the broker. With the default value of `-1`, the client will keep one such request pending at all times, ready to pull messages as fast as possible. With a large number of clients, this can cause a heavy drain on server resources, causing the server to become unresponsive. Setting the `pullPeriod` property to a positive value configures the client’s HTTP/HTTPS transport driver to wait that many seconds between pull requests, conserving server resources at the expense of increased response times to clients.

The `connectionTimeout` property specifies the interval, in seconds, that the client runtime waits for a response from the HTTP/HTTPS tunnel servlet before throwing an exception, as well as the time the broker waits after communicating with the tunnel servlet before freeing a connection. (A timeout is necessary in this case because the broker and the tunnel servlet have no way of knowing if a client that is accessing the tunnel servlet has terminated abnormally.)

Step 6 (HTTP and HTTPS): Configuring a Connection

To make HTTP/HTTPS connections to a broker, a client application needs an appropriately configured connection factory administered object. Before configuring the connection factory, clients wishing to use secure HTTPS connections must also have access to SSL libraries provided by the Java Secure Socket Extension (JSSE) and must obtain a trusted root certificate.

Installing a Root Certificate (HTTPS Only)

If the root certificate of the certification authority (CA) that signed your application server's (or Web server's) certificate is not in the trust store by default, or if you are using a proprietary application server or Web server certificate, you must install the root certificate in the trust store. (This step is not needed for ordinary, non-secure HTTP connections, or if the CA's root certificate is already in the trust store by default.)

▼ Installing a Root Certificate in the Trust Store

1 Import the root certificate.

Execute the command

```
JRE_HOME/bin/keytool -import -trustcacerts
                        -alias certAlias -file certFile
                        -keystore trustStoreFile
```

where *certFile* is the file containing the root certificate, *certAlias* is the alias representing the certificate, and *trustStoreFile* is the file containing your trust store.

2 Confirm that you trust the certificate.

Answer YES to the question Trust this certificate?

3 Identify the trust store to the client application.

In the command that launches the client application, use the `-D` option to specify the following properties:

```
javax.net.ssl.trustStore=trustStoreFile
javax.net.ssl.trustStorePassword=trustStorePassword
```

Configuring the Connection Factory (HTTP and HTTPS)

To enable HTTP/HTTPS support, you need to set the connection factory's `imqAddressList` attribute to the URL of the HTTP/HTTPS tunnel servlet. The URL has the form

```
http://hostName:portNumber/contextRoot/tunnel
```

or

```
https://hostName:portNumber/contextRoot/tunnel
```

where *hostName:portNumber* is the host name and port number of the application server or Web server hosting the tunnel servlet and *contextRoot* is the context root directory you specified when deploying the tunnel servlet on the server, as described above under [“Step 4 \(HTTP and HTTPS\): Deploying the Tunnel Servlet” on page 392](#).

You can set the `imqAddressList` attribute in any of the following ways:

- Use the `-o` option to the `imqobjmgr` command that creates the connection factory administered object (see “[Adding a Connection Factory](#)” on page 201).
- Set the attribute when creating the connection factory administered object using the Administration Console (`imqadmin`).
- Use the `-D` option to the command that launches the client application.
- Use an API call to set the attributes of the connection factory after you create it programmatically in client application code (see the *Message Queue Developer's Guide for Java Clients*).

Using a Single Servlet to Access Multiple Brokers (HTTP and HTTPS)

It is not necessary to configure multiple application or Web servers and tunnel servlets in order to access multiple brokers; you can share a single server instance and tunnel servlet among them. To do this, you must configure the `imqAddressList` connection factory attribute as follows:

```
http://hostName:portNumber/contextRoot/tunnel?ServerName=brokerHostName:instanceName
```

or

```
https://hostName:portNumber/contextRoot/tunnel?ServerName=brokerHostName:instanceName
```

where *brokerHostName* is the broker instance host name and *instanceName* is the name of the specific broker instance you want your client to access.

To check that you have entered the correct values for *brokerHostName* and *instanceName*, generate a status report for the HTTP/HTTPS tunnel servlet by accessing the servlet URL from a browser:

```
http://localhost:8080/imqhttp/tunnel
```

The report lists all brokers being accessed by the servlet, as shown in [Example C-1](#).

EXAMPLE C-1 Tunnel Servlet Status Report

```
HTTP tunnel servlet ready.
Servlet Start Time : Thu May 30 01:08:18 PDT 2002
Accepting secured connections from brokers on port : 7675
Total available brokers = 2
Broker List :
    helios:broker1
    selene:broker2
```

Using an HTTP Proxy

To use an HTTP proxy to access the HTTPS tunnel servlet, set the system properties `http.proxyHost` and `http.proxyPort` to the proxy server's host name and port number. You can set these properties using the `-D` option to the command that launches the client application.

Troubleshooting

This section describes possible problems with an HTTP or HTTPS connection and provides guidance on how to handle them.

Server or Broker Failure

The consequences of a server or broker failure in an (HTTP or HTTPS) connection vary depending on the specific component that has failed:

- If the application server or Web server fails and is restarted, all existing connections are restored with no effect on clients.
- If the broker fails and is restarted, an exception is thrown and clients must reestablish their connections.
- In the unlikely event that both the broker and the application server or Web server fail and the broker is not restarted, the application server or Web server will restore client connections and continue waiting for a broker connection without notifying clients. To avoid this situation, always restart the broker after a failure.

Client Failure to Connect Through the Tunnel Servlet

If an HTTPS client cannot connect to the broker through the tunnel servlet, do the following:

▼ If a Client Cannot Connect

- 1 Start the tunnel servlet and the broker.
- 2 Use a browser to access the servlet manually through the tunnel servlet URL.
- 3 Use the following administrative commands to pause and resume the connection:

```
imqcmd pause svc -n httpsjms -u admin
imqcmd resume svc -n httpsjms -u admin
```

When the service resumes, an HTTPS client should be able to connect to the broker through the tunnel servlet.

JMX Support

Message Queue includes support for Java-based client programs to programmatically configure and monitor Message Queue resources by means of the Java Management Extensions (JMX) application programming interface. These resources include brokers, connection services, connections, destinations, durable subscribers, and transactions. Use of the JMX API from the client side is fully described in the *Message Queue Developer's Guide for JMX Clients*. This appendix describes the JMX support infrastructure on the broker side, including the following topics:

- [“JMX Connection Infrastructure” on page 399](#)
- [“JMX Configuration” on page 402](#)

JMX Connection Infrastructure

The JMX API allows Java client applications to monitor and manage broker resources by programmatically accessing JMX MBeans (*managed beans*) that represent broker resources. As explained in the [“JMX-Based Administration” in *Sun Java System Message Queue 4.2 Technical Overview*](#), the broker implements MBeans associated with individual broker resources, such as connection services, connections, destinations, and so forth, as well as with whole categories of resources, such as the set of all destinations on a broker. There are separate *configuration MBeans* and *monitor MBeans* for setting a resource's configuration properties and monitoring its runtime state.

MBean Access Mechanism

In the JMX implementation used by Message Queue, JMX client applications access MBeans using remote method invocation (RMI) protocols provided by JDK 1.5 (and later).

When a broker is started, it automatically creates MBeans that correspond to broker resources and places them in an MBean server (a container for MBeans). JMX client applications access

the MBean server by means of an *JMX RMI connector* (heretofore called a JMX connector), which is used to obtain an *MBean server connection*, which, in turn, provides access to individual MBeans.

The broker also creates and configures two default JMX connectors, `jmxrmi` and `ssljmxrmi`. These connectors are similar to the broker connection services used to provide connections to the broker from JMS clients. By default, only the `jmxrmi` connector is activated at broker startup. The `ssljmxrmi` connector, which is configured to use SSL encryption, can be activated using the `imq.jmx.connector.activelist` broker property (see [“To Activate the SSL-Based JMX connector”](#) on page 406).

JMX client applications programmatically access JMX MBeans by first obtaining an MBean server connection from the `jmxrmi` or `ssljmxrmi` connector. The connector itself is accessed by using a proxy object (or stub) that is obtained from the broker by the JMX client runtime, as shown in the following figure. Encapsulated in the connector stub is the port at which the connector resides, which is dynamically assigned each time a broker is started, and other connection properties.

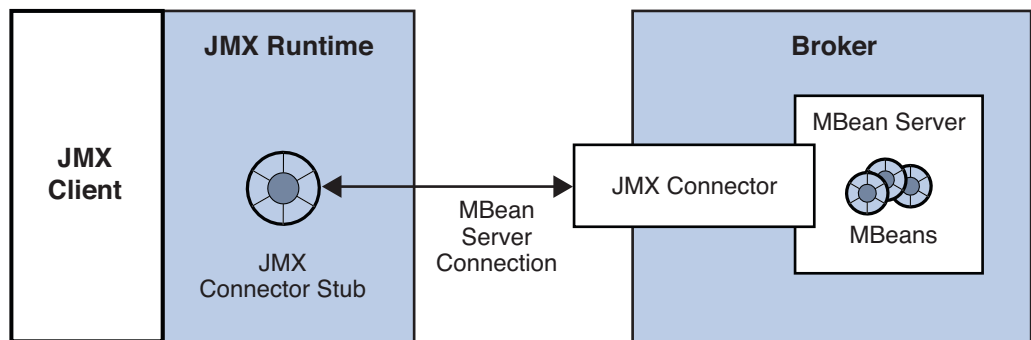


FIGURE D-1 Basic JMX Infrastructure

The JMX Service URL

JMX client applications obtain a JMX connector stub using an address called the JMX service URL. The value and format of the JMX service URL depends on how the broker's JMX support is configured:

- **Static JMX service URL.** The JMX service URL specifies the location of the JMX connector stub in an RMI registry. When the broker is started, it creates the JMX connector stub and places it in the specified location in the RMI registry. This location is fixed across broker startups.
- **Dynamic JMX service URL.** The JMX service URL contains the JMX connector stub as a serialized object. This URL is dynamically created each time the broker is started.

A JMX service URL has the following form:

```
service:jmx:rmi://brokerHost[:connectorPort]urlpath
```

where `rmi://brokerHost[:connectorPort]` specifies the host (and optionally a port) used by the JMX connector. By default the port is assigned dynamically on broker startup, but can be set to a fixed value for JMX connections through a firewall.

The *urlpath* portion of the JMX service URL depends on whether the JMX service URL is static (see [“Static JMX Service URL: Using an RMI Registry” on page 404](#)) or dynamic (see [“Dynamic JMX Service URL: Not Using an RMI Registry” on page 405](#)). In either case, you can determine the value of the JMX service URL by using the `imqcmd list jmx` subcommand (see the examples in [“RMI Registry Configuration” on page 402](#)).

By default, the broker does not use an RMI registry, and the JMX runtime obtains a JMX connector stub by extracting it from a dynamic JMX service URL. However, if the broker is configured to use an RMI registry, then JMX runtime uses a static JMX service URL to perform a JNDI lookup of the JMX connector stub in the RMI registry. This approach, illustrated in the following figure, has the advantage of providing a fixed location at which the connector stub resides, one that does not change across broker startups.

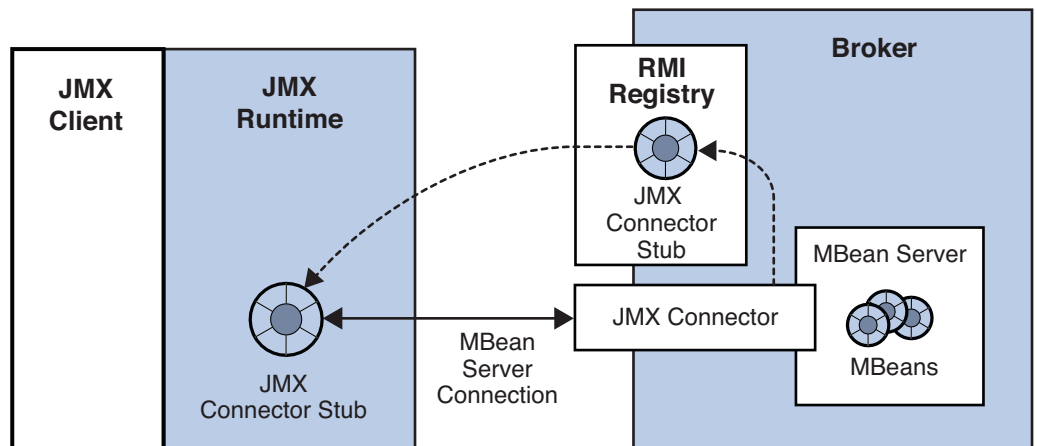


FIGURE D-2 Obtaining a Connector Stub from an RMI Registry

The Admin Connection Factory

Message Queue also provides, as a convenience, an `AdminConnectionFactory` class that hides the details of the JMX Service URL and JMX connector stub. The Admin Connection Factory uses the Message Queue Port Mapper service to get the relevant JMX Service URL (regardless of

the form being used) and thereby obtain a JMX connector stub. JMX applications that use the Admin Connection Factory only need to know the broker's host and Port Mapper port. The scheme is shown in the following figure.

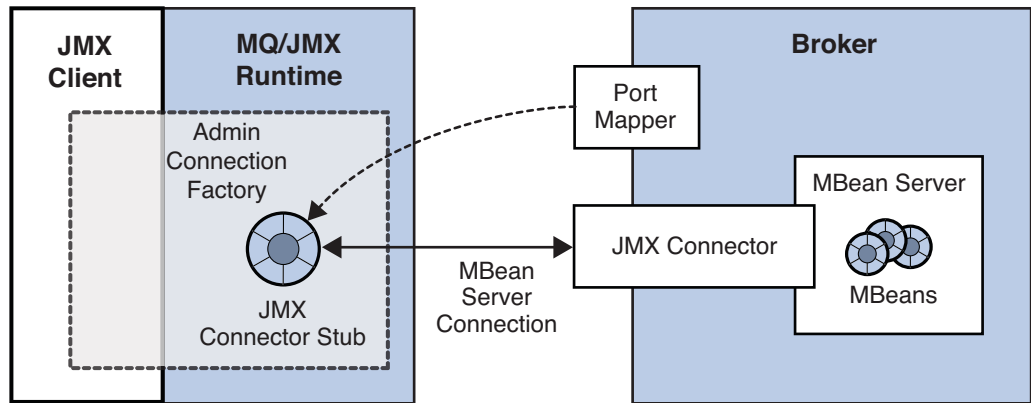


FIGURE D-3 Obtaining a Connector Stub from an Admin Connection Factory

For programmatic details, see [“Obtaining a JMX Connector from an Admin Connection Factory”](#) in *Sun Java System Message Queue 4.2 Developer’s Guide for JMX Clients*

JMX Configuration

Broker configuration properties that support JMX are listed in [Table 16-12](#). These properties can be set in the broker's instance configuration file (`config.properties`) or at broker startup with the `-D` option of the Broker utility (`imqbrokerd`). None of these properties can be set dynamically with the Command utility (`imqcmd`). In addition, as described below, some of these properties can be set with corresponding `imqbrokerd` options.

This section discusses several JMX configuration topics:

- [“RMI Registry Configuration”](#) on page 402
- [“SSL-Based JMX Connections”](#) on page 406
- [“JMX Connections Through a Firewall”](#) on page 407

RMI Registry Configuration

You can configure the broker to do any of the following:

- **Start an RMI registry** (`imq.jmx.rmiregistry.start=true`)

If the broker is configured to start an RMI registry, then the broker will do the following:

- Start an RMI registry in the broker process. The RMI registry will remain operational during the lifetime of the broker.
- Store the JMX connector stub for it's connectors in this RMI registry.
- Advertise a static JMX Service URL that points to the relevant JMX connector stub in this registry.
- Shut down the RMI registry as part of the broker shutdown process.
- **Use an existing RMI registry** (`imq.jmx.rmiregistry.use=true`)
 If the broker is configured to use an existing RMI registry on the local host, then the broker will do the following:
 - Expect an RMI registry to be running on the same host (at a port which can also be specified)
 - Store the JMX connector stub for it's connectors in this externally managed RMI registry.
 - Advertise a static JMX Service URL that points to the relevant JMX connector stub in this registry. This means the registry must remain operational during the lifetime of the broker.
 - Not shut down the RMI registry as part of the broker shutdown process.
- **Not use a registry at all** (both `imq.jmx.rmiregistry.start` and `imq.jmx.rmiregistry.use` are set to `false`).
 If the broker is configured to not use a registry, then the broker will advertise a dynamic JMX Service URL that contains the JMX connector stub as a serialized object.

The choice of using or not using an RMI registry depends upon whether you want a static or dynamic JMX Service URL, respectively. The advantages and disadvantages of using an RMI registry are shown in the following table.

TABLE D-1 Advantages and Disadvantages of Using an RMI Registry

Scenario	Broker Configuration	Advantages	Disadvantages
Using a Registry (Static JMX Service URL)	Configuration Properties: <code>imq.jmx.rmiregistry.start</code> <code>imq.jmx.rmiregistry.use</code> <code>imq.jmx.rmiregistry.port</code>	The value of the JMX Service URL is constant across broker restarts.	Broker depends on an RMI registry, either one it starts or one that is externally available. There is therefore one more port to worry about with regard to port conflicts or firewall configurations.

TABLE D-1 Advantages and Disadvantages of Using an RMI Registry (Continued)			
Scenario	Broker Configuration	Advantages	Disadvantages
Not Using a Registry (Dynamic JMX Service URL)	Default	Broker does not start up an RMI registry. There is therefore one less port to worry about with regard to port conflicts or firewall configurations.	The value of the JMX Service URL changes at every broker startup. JMX applications need to be provided a new URL every time the broker restarts. (This is not an issue with JMX client applications that use the AdminConnectionFactory class.)

If a registry is being used, the `imq.jmx.rmi.registry.port` property specifies the port number for the RMI registry. For convenience, you can also specify these RMI registry related properties by using equivalent Broker utility (`imqbrokerd`) options at broker startup: `-startRmiRegistry`, `-useRmiRegistry`, and `-rmiRegistryPort`, respectively (see [Table 15-1](#)).

Static JMX Service URL: Using an RMI Registry

When using an RMI Registry to store a JMX connector stub, the `url/path` portion of the JMX service URL (see “[The JMX Service URL](#)” on page 400) does not change across broker startups and has the following form:

`/jndi/rmi://brokerHost[:rmiPort]/brokerHost/portMapperPort/connectorName`

This path consists of two segments:

- `/jndi/rmi://brokerHost[:rmiPort]`
Specifies the RMI registry host and port at which the JMX contector stub is obtained by performing a JNDI lookup. The default port is 1099.
- `/brokerHost/portMapperPort/connectorName`
Specifies the location within the RMI registry where the JMX connector stub is stored.

EXAMPLE D-1 JMX Service URL When Using an RMI Registry

The following example shows the JMX service URL for the default `jmx.rmi` connector in the case where an RMI registry is started on port 1098 on a host called `yourhost`:

```
# imqbrokerd -startRmiRegistry -rmiRegistryPort 1098

% imqcmd list jmx -u admin -passfile /myDir/psswds
Listing JMX Connectors on the broker specified by:
```

Host	Primary Port

EXAMPLE D-1 JMX Service URL When Using an RMI Registry (Continued)

```
localhost      7676
```

```
Name      Active URL
jmxrmi     true    service:jmx:rmi://yourhost/jndi/rmi://yourhost:1098
              /yourhost/7676/jmxrmi
ssljmxrmi  false
```

Successfully listed JMX Connectors.

The JMX service URL could potentially contain a hostname and port three separate times, indicating the location of the JMX connector, the RMI registry, and the broker, respectively.

Dynamic JMX Service URL: Not Using an RMI Registry

When not using an RMI Registry to store a JMX connector stub, the *urlpath* portion of the JMX service URL is dynamically generated at broker startup and has the following form:

```
/stub/r00ABdmVyLlJlpIDJyGvQkwAAARod97VdgAEaA==
```

where the string following `/stub/` is the is the serialized JMX connector stub encoded in BASE64 (shortened above for legibility)

EXAMPLE D-2 JMX Service URL When Not Using an RMI Registry

The following example shows the JMX service URL for the default `jmxrmi` connector when no RMI registry is started by the broker and no existing registry is used.

```
# imqbrokerd
```

```
% imqcmd list jmx -u admin -passfile /myDir/psswds
Listing JMX Connectors on the broker specified by:
```

```
-----
Host      Primary Port
-----
localhost  7676

Name      Active URL
jmxrmi     true    service:jmx:rmi://yourhost/stub/r00ABdmVyLlJlpIDJy==
ssljmxrmi  false
```

Successfully listed JMX Connectors.

SSL-Based JMX Connections

If you need to have secure, encrypted connections between a JMX client and the broker's MBean server, then you need to configure both sides of the connection accordingly.

Broker Side SSL Configuration

As mentioned in “[JMX Connection Infrastructure](#)” on page 399, a broker is configured by default for non-secure communication using the preconfigured `jmxrmi` connector. Applications wishing to use the Secure Socket Layer (SSL) for secure communication must activate the alternate `ssljmxrmi` connector. The `ssljmxrmi` connector is preconfigured with `imq.jmx.connector.RMIconnectorName.useSSL=true`.

▼ To Activate the SSL-Based JMX connector

- 1 **Obtain and install a signed certificate.**

The procedure is the same as for the `ssljms`, `ssladmin`, or `cluster` connection service, as described under “[Using Signed Certificates](#)” on page 165.

- 2 **Install the root certification authority certificate in the trust store if necessary.**

- 3 **Add the `ssljmxrmi` connector to the list of JMX connectors to be activated at broker startup:**

```
imq.jmx.connector.activelist=jmxrmi,ssljmxrmi
```

- 4 **Start the broker.**

Use the Broker utility (`imqbrokerd`), either passing it the keystore password in a passfile or typing it from at the command line when prompted.

- 5 **Disable validation of certificates if desired.**

By default, the `ssljmxrmi` connector (or any other SSL-based connector) is configured to validate all broker SSL certificates presented to it. Validation will fail if the signer of the certificate is not in the client's trust store. To avoid this validation (for instance, when using self-signed certificates during software testing), set the broker property `imq.jmx.connector.ssljmxrmi.brokerHostTrusted` to `true`.

JMX Client Side SSL Configuration

On the client side, if the `AdminConnectionFactory` class is being used to obtain a JMX connector, the `AdminConnectionFactory` object must be configured with a URL specifying the `ssljmxrmi` connector:

```
AdminConnectionFactory acf = new AdminConnectionFactory();
acf.setProperty(AdminConnectionFactoryConfiguration.imqAddress,
    "mq://myhost:7676/ssljmxrmi");
```

In addition, if the JMX client needs to access the trust store, use the system properties `javax.net.ssl.trustStore` and `javax.net.ssl.trustStorePassword` to point the JMX client to the trust store. For example:

```
java -Djavax.net.ssl.trustStore=/tmp/myTruststore
-Djavax.net.ssl.trustStorePassword=myTrustword MyApp
```

JMX Connections Through a Firewall

If a JMX client application needs to connect to a broker that is located behind a firewall, the broker must be configured to use fixed JMX ports so the firewall can, in turn, be configured to allow traffic on these ports. The relevant ports are the following:

- The port used by the JMX connector. The property used to configure this port is `imq.jmx.connector.connectorName.port`, where *connectorName* can be `jmxrmi` or `ssljmxrmi`.
- The port used by the RMI registry, if any. The property used to configure this port is `imq.jmx.rmiregistry.port`. The equivalent command line option for `imqbrokerd` is `-rmiRegistryPort`.

Once these ports are specified, configure the firewall to allow traffic on these ports.

EXAMPLE D-3 JMX Configuration for Firewall When Not Using a RMI Registry

The following example starts a broker with no RMI registry and a `jmxrmi` connector on port 5656 on a host called `yourhost`, as follows:

```
# imqbrokerd -Dimq.jmx.connector.jmxrmi.port=5656
```

The resulting JMX service URL is:

```
service:jmx:rmi://yourhost:5656/stub/r00ABdmVyLlJlpIDJy==
```

The JMX service URL shows the connector port. In this case, you need to configure the firewall to allow traffic only on port 5656.

EXAMPLE D-4 JMX Configuration for Firewall When Using an RMI Registry

The following example starts a broker with an RMI registry on port 1098 and a `jmxrmi` connector on port 5656 on a host called `yourhost`, as follows:

```
# imqbrokerd -startRmiRegistry -rmiRegistryPort 1098
-Dimq.jmx.connector.jmxrmi.port=5656
```

The resulting JMX service URL is:

EXAMPLE D-4 JMX Configuration for Firewall When Using an RMI Registry *(Continued)*

```
service:jmx:rmi://yourhost:5656/jndi/rmi://yourhost:1098  
/yourhost/7676/jmxrmi
```

The JMX service URL shows both these ports. You need to configure the firewall to allow traffic on ports 1098 and 5656.

Frequently Used Command Utility Commands

This appendix lists some frequently used Message Queue Command utility (`imqcmd`) commands. For a comprehensive list of command options and attributes available to you from the command line, refer to “[Command Utility](#)” on page 282 in “[Command Utility](#)” on page 282

Syntax

```
imqcmd subcommand argument [
options]
imqcmd -h|H
imqcmd -v
```

-H or -h provides comprehensive help. The -v subcommand provides version information.

When you use `imqcmd`, the Command utility prompts you for a password. To avoid the prompt (and to increase security), you can use the -passfile *pathToPassfile* option to point the utility to a password file that contains the administrator user name and password.

Example: `imqcmd query bkr -u adminUserName -passfile pathToPassfile -b myServer:7676`

Broker and Cluster Management

```
imqcmd query bkr
imqcmd pause bkr
imqcmd restart bkr
imqcmd resume bkr
imqcmd shutdown bkr -b myBroker:7676
imqcmd update bkr -o "imq.system.max_count=1000"
imqcmd reload cls
```

Broker Configuration Properties (-o option)

“[Broker Configuration Properties \(-o option\)](#)” on page 410 lists frequently used broker configuration properties. For a full list of broker configuration properties and their descriptions, see [Chapter 16, “Broker Properties Reference”](#)

TABLE E-1 Broker Configuration Properties (-o option)

Property	Notes
imq.autocreate.queue	
imq.autocreate.queue.maxNumActiveConsumers	Specify -1 for unlimited
imq.autocreate.queue.maxNumBackupConsumers	Specify -1 for unlimited
imq.autocreate.topic	
imq.cluster.url	
imq.destination.DMQ.truncateBody	
imq.destination.logDeadMessages	
imq.log.file.rolloverbytes	Specify -1 for unlimited
imq.log.file.rolloversecs	Specify -1 for unlimited
imq.log.level	NONE ERROR WARNING INFO
imq.message.max_size	Specify -1 for unlimited
imq.portmapper.port	
imq.system.max_count	Specify -1 for unlimited
imq.system.max_size	Specify -1 for unlimited

Service and Connection Management

```
imqcmd list svc
imqcmd query svc
imqcmd update svc -n jms -o "minThreads=200" -o "maxThreads=400" -o "port=8995"
imqcmd pause svc -n jms
imqcmd resume svc -n jms
imqcmd list cxn -svn jms
imqcmd query cxn -n 1234567890
```

Durable Subscriber Management

```
imqcmd list dur -d MyTopic
imqcmd destroy dur -n myDurSub -c "clientID-111.222.333.444"
imqcmd purge dur -n myDurSub -c "clientID-111.222.333.444"
```

Transaction Management

```
imqcmd list txn
imqcmd commit txn -n 1234567890
imqcmd query txn -n 1234567890
imqcmd rollback txn -n 1234567890
```

Destination Management

```
imqcmd create dst -n MyQueue -t q -o "maxNumMsgs=1000" -o "maxNumProducers=5"
imqcmd update dst -n MyTopic -t t -o "limitBehavior=FLOW_CONTROL| REMOVE_OLDEST|REJECT_NEWEST|REMOVE_LOW_PRIORITY"
imqcmd compact dst -n MyQueue -t q
imqcmd purge dst -n MyQueue -t q
imqcmd pause dst -n MyQueue -t q -pst PRODUCERS|CONSUMERS|ALL
imqcmd resume dst -n MyQueue -t q
imqcmd destroy dst -n MyQueue -t q
imqcmd query dst -n MyQueue -t q
imqcmd list dst -tmp
```

Destination Configuration Properties (-o option)

“[Destination Configuration Properties \(-o option\)](#)” on page 411 lists frequently used destination configuration properties. For a full list of destination configuration properties and their descriptions, see [Chapter 17, “Physical Destination Property Reference”](#)

TABLE E-2 Destination Configuration Properties (-o option)

Property	Notes
consumerFlowLimit	Specify -1 for unlimited
isLocalOnly (create only)	
limitBehavior	FLOW_CONTROL REMOVE_OLDEST REJECT_NEWEST REMOVE_LOW_PRIORITY

TABLE E-2 Destination Configuration Properties (-o option) (Continued)

Property	Notes
localDeliveryPreferred (queue only)	
maxNumActiveConsumers (queue only)	Specify -1 for unlimited
maxNumBackupConsumers (queue only)	Specify -1 for unlimited
maxBytesPerMsg	Specify -1 for unlimited
maxNumMsgs	Specify -1 for unlimited
maxNumProducers	Specify -1 for unlimited
maxTotalMsgBytes	Specify -1 for unlimited
useDMQ	

Metrics

```
imqcmd metrics bkr -m cxn|rts|ttl -int 5 -msp 20
imqcmd metrics svc -m cxn|rts|ttl
imqcmd metrics dst -m con|dsk|rts|ttl
```

Index

A

- access control file
 - location, 374, 375, 377
- acknowledgeMode ActivationSpec property, 352
- ActivationSpec JavaBean, 351
- addressList ActivationSpec property, 351
- addressList managed connection factory attribute, 350
- addressList Resource Adapter attribute, 348
- addressListBehavior managed connection factory attribute, 350
- addressListBehavior Resource Adapter attribute, 348
- addressListIterations managed connection factory attribute, 350
- addressListIterations Resource Adapter attribute, 348
- admin connection service, 95
- ADMIN service type, 94
- admin user
 - changing password, 144
 - initial entry, 140
- administered objects
 - attributes (reference), 337
 - deleting, 203
 - listing, 203-204
 - managing, 189-207
 - object store
 - See* object stores
 - querying, 204
 - required information, 199
 - updating, 205
 - XA connection factory
 - See* connection factory administered objects
- Administration Console
 - starting, 40
 - tutorial, 39
- administration tasks
 - development environment, 33-34
 - production environment, 34-35
- administration tools
 - Administration Console, 37
 - built-in, 36-37
 - command line utilities, 36
 - JMX-based administration, 38
- administrator password, 144
- API documentation, 374, 375, 377
- attributes of physical destinations, 333-336
- authentication
 - See also* access control
 - about, 137
 - JAAS-based
 - See* JAAS-based authentication
 - managing, 139-153
- authorization
 - See also* access control
 - about, 138
 - managing, 153-159
 - user groups, 138
- auto-create destinations, access control settings, 138
- auto-created destinations
 - access control settings, 158
 - broker properties (table), 303-307
 - default property values, 110
- automatic reconnection, 194
 - limitations, 194

AUTOSTART property, 69, 71

B

benchmarks, performance, 230-231

bottlenecks, performance, 233

broker clusters

- adding brokers to, 180

- architecture, 241

- configuration file, 172, 178, 179, 184, 321

- configuration properties, 171, 321-324

- connecting conventional brokers, 178

- conventional

 - automatic reconnection in, 194

- high-availability

 - automatic reconnection in, 194

- listing brokers, 92, 176-177, 287

- pausing physical destinations in, 111

- performance effect of, 241

- purging physical destinations in, 112

- reasons for using, 241

- reloading configuration, 287

- secure interbroker connections (SSL), 180

broker components

- clustering services, 78

- connection services, 77, 93

- monitoring services, 78, 209-210

- persistence services, 77, 125-126

- routing services, 77, 119

- security services, 77, 135-138

broker metrics

- Logger properties, 213, 217, 320

- metrics messages, 224

- metrics quantities (table), 356-358

- reporting interval, Logger, 281

- using broker log files, 217

- using imqcmd, 220-221, 222

- using message-based monitoring, 225

broker monitoring service, properties, 317-321

broker responses, wait period for client, 342

broker states, 176

brokers

- access control

 - See* authorization

brokers (*Continued*)

- auto-create physical destination properties, 303-307

- clock synchronization, 67-68

- clustering, 178

- clusters

 - See* broker clusters

- commands for managing, 285-288

- configuration files

 - See* configuration files

- displaying metrics, 287

- failure recovery, 125

- httpjms connection service properties, 394

- httpsjms connection service properties, 394

- instance configuration properties, 79

- instance name, 278

- interconnected

 - See* broker clusters

- JMX, and, 38, 223

- limit behaviors, 120, 241

- listing, 92

- listing connection services, 99

- listing property values, 287

- logging

 - See* Logger

- managing, 83-92

- memory management, 120, 241

- message capacity, 90, 120, 302

- message flow control

 - See* message flow controls

- metrics

 - See* broker metrics

- pausing, 89, 286

- permissions required for starting, 68

- programmatic management of, 38, 223

- properties (reference), 299, 333

- querying, 91

- quiescing, 88, 286

- re-creating state, 125

- removing, 74

- restarting, 88, 286

- restarting automatically, 69, 71

- resuming, 89, 286

- running as Windows service, 71-74

- setting properties, 287

brokers (*Continued*)

- shutting down, 87, 286
- starting automatically, 69-74
- starting interactively, 68-69
- startup with SSL, 163
- states of
 - See* broker states
- takeover, 287
- unquiescing, 88, 286
- updating properties of, 89-90
- viewing information about, 91-92
- viewing metric information, 92

C

certificates

- self-signed, 159-165, 385
- signed, 165-168

client applications

- example, 374, 375, 377
- factors affecting performance, 234-238

client identifier (ClientID)

- for durable subscribers, 196-197
- in destroying durable subscription, 122

client runtime

- configuration of, 242
- message flow tuning, 246

clientID activation specification attribute, 352

clientID ActivationSpec property, 352

clientID managed connection factory attribute, 350

clients

- clock synchronization, 67-68
- starting, 74-75

clock synchronization, 67-68

cluster configuration file, 172, 178, 179, 184, 321

cluster configuration properties, 171, 321

cluster connection service

- configuring for SSL, 159, 180
- host name or IP address for, 173, 322
- network transport for, 172, 322
- port number for, 173, 322

cluster identifier, 174, 323

cluster.properties file, 78

clustering brokers, 178

clustering services, broker, 78

clusters, *See* broker clusters

command files, 205-207

command line syntax, 277

command line utilities

- about, 36
- basic syntax, 277
- displaying version, 85, 142, 285
- help, 86, 142, 285
- imqbrokerd, *See*, imqbrokerd command, 36
- imqcmd, *See*, imqcmd command, 36
- imqdbmgr *See*, imqdbmgr command, 36
- imqkeytool, *See*, imqkeytool command, 36
- imqobjmgr, *See*, imqobjmgr command, 36
- imqsvcadm, *See*, imqsvcadm command, 36
- imqusermgr, *See*, imqusermgr command, 36

command line utility executables

- location, 373, 375, 376

command options, as configuration overrides, 75

compacting

- file-based data store, 127
- physical destinations, 117

config.properties file, 78, 79, 180, 181, 394

configuration change record

- backing up, 182
- restoring, 183

configuration files

- broker (figure), 79
- cluster, 78, 172, 178, 179, 184, 321
- default, 78
- installation, 78
- instance, 78, 79, 373, 375, 376
- location, 373, 375, 376
- modifying, 78
- template location, 373, 375, 376
- templates, 373, 375, 376

connection factories, adding administered objects

- for, 201

connection factory administered objects

- application server support attributes, 344
- attributes, 192-199
- client identification attributes, 195-197
- connection handling attributes, 192-195
- JMS properties support attributes, 198

- connection factory administered objects (*Continued*)
 - overriding message header fields, 198
 - queue browser behavior attributes, 198, 343-344
 - reliability and flow control attributes, 197
 - standard message properties, 344
- connection service metrics
 - metrics quantities, 358-359
 - using imqcmd metrics, 221
 - using imqcmd query, 222
- connection services
 - access control for, 136, 311
 - activated at startup, 300
 - admin
 - See* admin connection service
 - cluster, 322
 - See* cluster connection service
 - commands for managing, 288
 - displaying metrics, 288
 - HTTP
 - See* HTTP connections
 - httpjms
 - See* httpjms connection service
 - HTTPS
 - See* HTTPS connections
 - httpsjms
 - See* httpsjms connection service
 - jms
 - See* jms connection service
 - listing, 99
 - listing available, 288
 - listing property values, 288
 - pausing, 97, 288
 - Port Mapper
 - See* Port Mapper
 - properties, 300-302
 - protocol type, 94
 - querying, 99
 - resuming, 98, 288
 - service type, 94
 - SSL-based, 162
 - ssladmin
 - See* ssladmin connection service
 - ssljms
 - See* ssljms connection service

- connection services (*Continued*)
 - ssljms connection service, 94
 - thread allocation, 98
 - thread pool management, 96
 - updating, 98, 288
 - viewing information about, 99-101
 - viewing metric information, 100
- connection services, broker, 77
- connections
 - automatic reconnection
 - See* automatic reconnection
 - commands for managing, 288-289
 - destroying, 102, 289
 - failover
 - See* automatic reconnection
 - limited by file descriptor limits, 68
 - listing, 101, 289
 - performance effect of, 239-240
 - querying, 102, 123, 289
- connectionURL Resource Adapter attribute, 348
- consumerFlowLimit destination property, 335, 369
- consumerFlowLimit property, 197
- customAcknowledgeMode ActivationSpec
 - property, 352

D

- data store
 - about, 125
 - compacting, 127
 - configuring, 126
 - contents of, 125
 - file-based, 126-127
 - JDBC-based, 128-130
 - location, 374, 375, 376
 - performance effect of, 241-242
 - resetting, 279
 - synchronizing to disk, 127
- dead message queue
 - configuring use of, 118
 - described, 118-119
 - logging, 119, 213
 - managing, 118-119
 - truncating message bodies, 119

dead message queue (*Continued*)
 UseDMQ property, 369
 variant treatment of physical destination
 properties, 118-119

dead messages
 See also dead message queue
 logging, 213

default.properties file, 78

deleting, broker instance, 74

delivery modes, performance effect of, 235

destination activation specification attribute, 351

destination ActivatioSpec property, 351

destination administered objects, attributes, 199

destination metrics
 metrics quantities, 359-364
 using imqcmd metrics, 219, 221-222
 using imqcmd query, 222
 using message-based monitoring, 225

destinations
 See physical destinations
 adding administered objects for, 201-203

destinationType activation specification attribute, 352

destinationType ActivationSpec property, 351

destroying
 connections, 102, 289
 durable subscriptions, 291
 physical destinations, 110, 289

development environment administration tasks, 33-34

directory lookup for clusters (Linux), 178

directory variables
 IMQ_HOME, 25
 IMQ_JAVAHOME, 26
 IMQ_VARHOME, 26

disk utilization by physical destinations, 116-118

displaying product version, 85, 142, 285

DN username format, 315

durable subscriptions
 commands for managing, 291
 destroying, 122, 291
 listing, 121, 291
 managing, 121
 performance effect of, 237
 purging messages for, 291

E

encryption
 about, 135, 138
 implementing, 159-168
 Key Tool and, 138

endpointExceptionRedeliveryAttempts activation
 specification attribute, 353

endpointExceptionRedeliveryAttempts ActivationSpec
 property, 352

environment variables, *See* directory variables

/etc/hosts file (Linux), 178

example applications, 374, 375, 377

external resource files directory, 374, 376, 377

F

file-based persistence, 126-127

file descriptor limits, 68
 connection limits and, 68

file synchronization
 imq.persist.file.sync.enabled option, 309
 with Sun Cluster, 309

firewalls, 383

flow control, *See* message flow controls

fragmentation of messages, 127

G

guest user, 140

H

hardware, performance effect of, 239

help (command line), 86, 142, 285

high-availability clusters, takeover states, 176

hosts file (Linux), 178

HTTP
 connection service
 See httpjms connection service
 proxy, 383
 support architecture, 383-384
 transport driver, 383

- HTTP connections
 - request interval, 394
 - support for, 383
 - tunnel servlet
 - See HTTP tunnel servlet
- HTTP protocol type, 95
- HTTP tunnel servlet
 - about, 383
 - deploying, 392-394
- httpjms connection service, 95
 - configuring broker for, 394-395
- HTTPS
 - connection service
 - See httpsjms connection service
 - support architecture, 383-384
- HTTPS connections
 - multiple brokers, for, 397-398
 - request interval, 394
 - support for, 383
 - tunnel servlet
 - See HTTPS tunnel servlet
- HTTPS protocol type, 95
- HTTPS tunnel servlet
 - about, 383
 - deploying, 392-394
- httpsjms connection service, 159
 - configuring broker for, 394-395
 - introduced, 95
 - setting up, 384-398
- I**
 - imq.accesscontrol.enabled property, 135, 311
 - imq.accesscontrol.file.filename property, 136, 312
 - imq.accesscontrol.file.url property, 312
 - imq.accesscontrol.type property, 311
 - imq.admin.tcp.port property, 170
 - imq.authentication.basic.user_repository property, 137, 311
 - imq.authentication.client.response.timeout property, 137, 311
 - imq.authentication.type property, 137, 311
 - imq.autocreate.destination.isLocalOnly property, 306
 - imq.autocreate.destination.limitBehavior property, 304, 305
 - imq.autocreate.destination.maxBytesPerMsg property, 304
 - imq.autocreate.destination.maxCount property, 304
 - imq.autocreate.destination.maxNumMsgs property, 304
 - imq.autocreate.destination.maxNumProducers property, 305
 - imq.autocreate.destination.maxTotalMsgBytes property, 304, 306
 - imq.autocreate.destination.useDMQ property, 118
 - imq.autocreate.queue.consumerFlowLimit property, 306
 - imq.autocreate.queue.localDeliveryPreferred property, 306
 - imq.autocreate.queue.maxNumActiveConsumers property, 90, 305
 - imq.autocreate.queue.maxNumBackupConsumers property, 90, 305
 - imq.autocreate.queue property, 90, 303
 - imq.autocreate.topic property, 90, 303
 - imq.broker.adminDefinedRoles.count property, 321, 327
 - imq.broker.adminDefinedRoles.nameN property, 321
 - imq.broker.adminDefinedRoles.nameN property, 327, 367
 - imq.brokerid property, 130, 174, 300
 - imq.cluster.brokerlist property, 173, 178, 179, 180, 181, 322
 - imq.cluster.clusterid property, 174, 323
 - imq.cluster.ha, 321
 - imq.cluster.ha property, 173
 - imq.cluster.heartbeat.hostname property, 175, 323
 - imq.cluster.heartbeat.interval property, 175, 323
 - imq.cluster.heartbeat.port property, 175, 323
 - imq.cluster.heartbeat.threshold property, 175, 323
 - imq.cluster.hostname property, 173, 322
 - imq.cluster.masterbroker property, 173, 180, 182, 323
 - imq.cluster.monitor.interval property, 175, 324
 - imq.cluster.monitor.threshold property, 175, 324
 - imq.cluster.port property, 173, 322
 - imq.cluster.transport property, 172, 180, 181, 322
 - imq.cluster.url property, 90, 172, 179, 180, 181, 321

- imq.destination.DMQ.truncateBody property, 90, 119, 120, 303
- imq.destination.logDeadMsgs property, 90, 213, 317
- IMQ_HOME directory variable, 25
- imq.hostname property, 96, 300
- imq.httpjms.http.connectionTimeout property, 394
- imq.httpjms.http.pullPeriod property, 394
- imq.httpjms.http.servletHost property, 394
- imq.httpjms.http.servletPort property, 394
- imq.httpsjms.https.connectionTimeout property, 394
- imq.httpsjms.https.pullPeriod property, 394
- imq.httpsjms.https.servletHost property, 394
- imq.httpsjms.https.servletPort property, 394
- imq.imqcmd.password property, 137, 169, 313
- IMQ_JAVAHOME directory variable, 26
- imq.jms.tcp.port property, 169
- imq.jmx.connector.activelist property, 324
- imq.jmx.connector.*RMIconnectorName*.brokerHostTrusted property, 325
- imq.jmx.connector.*RMIconnectorName*.port property, 325
- imq.jmx.connector.*RMIconnectorName*.urlpath property, 325
- imq.jmx.connector.*RMIconnectorName*.useSSL property, 325
- imq.jmx.rmiregistry.port property, 326
- imq.jmx.rmiregistry.start property, 326
- imq.jmx.rmiregistry.use property, 326
- imq.keystore.file.dirpath property, 162, 313
- imq.keystore.file.name property, 162, 313
- imq.keystore.password property, 138, 169, 313
- imq.log.console.output property, 213, 318
- imq.log.console.stream property, 213, 317
- imq.log.file.dirpath property, 213, 318
- imq.log.file.filename property, 213, 318
- imq.log.file.output property, 213, 318
- imq.log.file.rolloverbytes property, 90, 213, 318
- imq.log.file.rolloversecs property, 90, 213, 319
- imq.log.level property, 90, 213, 317
- imq.log.syslog.facility property, 319
- imq.log.syslog.identity property, 319
- imq.log.syslog.logconsole property, 320
- imq.log.syslog.logpid property, 319
- imq.log.syslog.output property, 213, 319
- imq.log.timezone property, 320
- imq.message.expiration.interval property, 120, 303
- imq.message.max_size property, 90, 120, 303
- imq.metrics.enabled property, 209, 320
- imq.metrics.interval property, 209, 320
- imq.metrics.topic.enabled property, 225, 320
- imq.metrics.topic.interval property, 225, 320
- imq.metrics.topic.persist property, 225, 320
- imq.metrics.topic.timetolive property, 225, 321
- imq.passfile.dirpath property, 137, 313
- imq.passfile.enabled property, 137, 313
- imq.passfile.name property, 137, 313
- imq.persist.file.destination.message.filepool.limit property, 127, 308
- imq.persist.file.message.cleanup property, 127, 308
- imq.persist.file.message.filepool.cleanratio property, 127, 308
- imq.persist.file.message.max_record_size property, 308
- imq.persist.file.message.vrfile.max_record_size property, 127
- imq.persist.file.sync.enabled property, 127, 309
- imq.persist.file.sync property, 127
- imq.persist.file.transaction.memorymappedfile.enabled property, 236, 309
- imq.persist.jdbc.dbVendor property, 129, 309
- imq.persist.jdbc.password property, 169
- imq.persist.jdbc.vendorName.closedburl property, 310
- imq.persist.jdbc.vendorName.createdburl property, 310
- imq.persist.jdbc.vendorName.driver property, 309
- imq.persist.jdbc.vendorName.needpassword property, 129, 310
- imq.persist.jdbc.vendorName.opendburl property, 309
- imq.persist.jdbc.vendorName.password property, 129, 310
- imq.persist.jdbc.vendorName.property.*propName*, 129
- imq.persist.jdbc.vendorName.property.*propName* property, 310
- imq.persist.jdbc.vendorName.tableoption property, 310
- imq.persist.jdbc.vendorName.user property, 129
- imq.persist.jdbc.vendorName.user property, 310

- imq.persist.store property, 125, 130, 307
- imq.ping.interval property, 97, 302
- imq.portmapper.backlog property, 96, 301
- imq.portmapper.hostname property, 96, 300
- imq.portmapper.port property, 90, 95, 300
- imq.primaryowner.contact property, 321, 330, 367
- imq.primaryowner.name property, 321, 330, 366
- imq.protocol.protocolType.inbufsz, 243
- imq.protocol.protocolType.nodelay, 243
- imq.protocol.protocolType.outbufsz, 243
- imq.resource_state.count property, 303
- imq.resource_state.threshold property, 303
- imq.resourceState.count property, 121
- imq.service.activelist property, 95, 300
- imq.service_name.accesscontrol.file.filename property, 312
- imq.service_name.accesscontrol.file.urlmax property, 312
- imq.service_name.authentication.type property, 311
- imq.service_name.max_threads property, 301, 368
- imq.service_name.min_threads property, 301, 368
- imq.service_name.protocol_type.hostname property, 300
- imq.service_name.protocol_type.port property, 301
- imq.service_name.threadpool_model property, 301
- imq.serviceName.accesscontrol.enabled property, 136, 312
- imq.serviceName.accesscontrol.file.filename property, 136
- imq.serviceName.authentication.type property, 137
- imq.serviceName.max_threads property, 96
- imq.serviceName.min_threads property, 96
- imq.serviceName.protocolType.hostname property, 96
- imq.serviceName.protocolType.port property, 95
- imq.serviceName.threadpool_model property, 96
- imq.shared.connectionMonitor_limit property, 97, 302
- imq.ssladmin.tls.port property, 170
- imq.ssljms.tls.port property, 169
- imq.system.max_count property, 90, 120, 302
- imq.system.max_size property, 90, 120, 302
- imq.transaction.autorollback property, 303
- imq.user_repository.jaas.groupPrincipalClass property, 316
- imq.user_repository.jaas.name property, 316
- imq.user_repository.jaas.userPrincipalClass property, 316
- imq.user_repository.ldap.base property, 314
- imq.user_repository.ldap.gidattr property, 315
- imq.user_repository.ldap.grpbase property, 315
- imq.user_repository.ldap.grpfilter property, 315
- imq.user_repository.ldap.grpsearch property, 315
- imq.user_repository.ldap.memattr property, 315
- imq.user_repository.ldap.password property, 137, 169, 314
- imq.user_repository.ldap.principal property, 137, 314
- imq.user_repository.ldap.property_name property, 314
- imq.user_repository.ldap.server property, 137, 313
- imq.user_repository.ldap.ssl.enabled property, 315
- imq.user_repository.ldap.timeout property, 315
- imq.user_repository.ldap.uidattr property, 314
- imq.user_repository.ldap.usrfilter property, 315
- imq.user_repository.ldap.usrformat property, 315
- IMQ_VARHOME directory variable, 26
- imqAckTimeout attribute, 197, 342
- imqAddressList attribute, 193, 194, 338
 - dynamically updated in high-availability clusters, 194, 338
- imqAddressListBehavior attribute, 193, 194, 338
- imqAddressListIterations attribute, 193, 194, 338
- imqbrokerd command, 68
 - about, 36
 - backing up configuration change record, 182
 - clearing the data store, 127
 - connecting brokers, 179
 - in password file, 168
 - options, 278-282
 - passing arguments to, 80
 - reference, 278
 - removing a broker, 74
 - removing a broker from a cluster, 182
 - restoring configuration change record, 183
 - setting logging properties, 215
- imqbrokerd.conf file, 69
- imqcmd command
 - about, 36
 - displaying version, 85

- imqcmd command (*Continued*)
 - general options, 284
 - in password file, 168
 - metrics monitoring, 218-222
 - physical destination management, 106-107
 - physical destination subcommands (table), 106-107
 - reference, 282
 - secure connection to broker, 164-165, 284
 - transaction management, 122
 - usage help, 86
- imqConfiguredClientID attribute, 196, 341
- imqConnectionFlowCount attribute, 197, 246, 342
- imqConnectionFlowLimit attribute, 197, 248, 342
- imqConnectionFlowLimitEnabled attribute, 197, 248, 342
- imqConsumerFlowLimit attribute, 197, 247, 343
- imqConsumerFlowThreshold attribute, 197, 247, 343
- imqdbmgr command
 - about, 36
 - in password file, 168
 - options, 295
 - reference, 294
- imqDefaultPassword attribute, 195, 196, 341
- imqDefaultUsername attribute, 195, 196, 341
- imqDestinationDescription attribute, 199, 345
- imqDestinationName attribute, 199, 345
- imqDisableSetClientID attribute, 341
- imqFlowControlLimit attribute, 343
- imqJMSDeliveryMode attribute, 345
- imqJMSExpiration attribute, 345
- imqJMSPriority attribute, 199, 345
- imqkeytool command
 - about, 36
 - reference, 298
 - using, 160
- imqLoadMaxToServerSession attribute, 198, 344
- imqobjmgr command
 - about, 36
 - options, 292
 - reference, 292
 - subcommands, 292
- imqOverrideJMSExpiration attribute, 345
- imqOverrideJMSDeliveryMode attribute, 345
- imqOverrideJMSHeadersToTemporaryDestinations attribute, 199, 345
- imqOverrideJMSPriority attribute, 199, 345
- imqPingInterval attribute, 195
- imqQueueBrowserMax MessagesPerRetrieve attribute, 198, 343
- imqQueueBrowserRetrieveTimeout attribute, 198, 343
- imqReconnectAttempts attribute, 194, 339
- imqReconnectEnabled attribute, 194, 339
- imqReconnectInterval attribute, 194, 339
- imqSetJMSXAppID attribute, 344
- imqSetJMSXConsumerTXID attribute, 344
- imqSetJMSXProducerTXID attribute, 344
- imqSetJMSXRcvTimestamp attribute, 344
- imqSetJMSXUserID attribute, 344
- imqSSLIsHostTrusted attribute, 339
- imqSSLIsHostTrusted attribute, 160
- imqsvcadm command
 - about, 36
 - options, 297
 - reference, 297
 - subcommands, 297
- imqusermgr command, 141-145
 - about, 36
 - displaying version, 142
 - general options, 296-297
 - general options (table), 142
 - options, 296
 - passwords, 142
 - reference, 295
 - subcommands, 296
 - subcommands (table), 141
 - usage help, 142
 - user names, 142
- imqusermgr utility, 136
- install.properties file, 78
- instance configuration files, *See* configuration files
- instance directory, removing, 74
- isLocalOnly destination property, 335, 369

J

- J2EE connector architecture (JCA), 347
- JAAS-based authentication, 148-153

JAAS-based authentication (*Continued*)

- configuration file, 148
- configuration file for, 150
- JAAS API, 148
- JAAS client, 148
- login module, 148
- setting up, 151-153

Java ES Monitoring Framework (JESMF), 223-224**Java Management Extensions**, *See* JMX**java.naming.factory.initial** attribute, 190, 191**java.naming.provider.url** attribute, 190, 191**java.naming.security.authentication** attribute, 191**java.naming.security.credentials** attribute, 190**java.naming.security.principal** attribute, 190**Java runtime**

- for Windows service, 72-73
- specifying path to, 280, 285, 293, 297

Java Virtual Machine, *See* JVM**javahome** option, 72**JCA (J2EE connector architecture)**, 347**JDBC-based persistence**

- about, 128-130
- JDBC driver, 309

JDBC-based persistence

- setting up, 130-132
- tuning for performance, 245

JDBC support, about, 128-130**JDBC support**, configuring, 126**JES Monitoring Framework (JESMF)**, 365-370

- brokers, 366-367
- common attributes, 365-366
- connection services, 367-368
- destinations, 369
- Message Queue product, 366
- persistent data store, 370
- Port Mapper, 367
- user repository, 370

JESMF, *See* JES Monitoring Framework**jms connection service**, 94**JMSDeliveryMode** message header field, 198**JMSExpiration** message header field, 198**JMSPriority** message header field, 198**JMSXAppID** message property, 198**JMSXConsumerTXID** message property, 198**JMSXProducerTXID** message property, 198**JMSXRcvTimestamp** message property, 198**JMSXUserID** message property, 198**JMX**

- administrative support for, 399-408
- commands for managing, 292
- configuration properties, 324

JMX-based administration, 38, 223**JMX connectors**, introduced, 400**JMX imqcmd** subcommands, 292**JNDI**

- lookup, 53
- lookup name, 199, 200
- object store, 36, 189
- object store attributes, 190-191, 200

jrehome option, 72**JVM****metrics***See* JVM metrics**performance effect of**, 239**tuning for performance**, 242-243**JVM metrics**

- metrics quantities, 356
- using broker log files, 217
- using imqcmd metrics, 220
- using message-based monitoring, 225

K**key pairs**

- generating, 161, 386
- regenerating, 162

key store, 162, 386**Key Tool**, 138**keystore**, file, 162**keytool** command

- command syntax, 385
- using, 385

L**LDAP repository**, 136

- LDAP server
 - as user repository, 145-147
 - object store attributes, 190
 - limit behaviors
 - broker, 120
 - physical destinations, 120, 334
 - limitBehavior destination property, 333, 334, 369
 - load-balanced queue delivery, *See* queue load-balanced delivery
 - localDeliveryPreferred destination property, 335, 369
 - log files
 - changing default location, 214
 - changing default name, 215
 - dead message logging, 218
 - default location, 374, 375, 376
 - names, 214
 - reporting metrics, 217
 - rollover criteria, 213, 216, 318
 - rollover frequency, 215
 - setting properties, 215
 - Logger
 - about, 212
 - changing configuration, 215
 - dead message format, 218
 - levels, 213, 281, 317
 - message format, 213
 - metrics information, 320
 - output channels, 212, 213, 215-216
 - redirecting log messages, 216
 - rollover criteria, 216
 - setting properties, 215
 - writing to console, 213, 282, 318
 - logging, *See* Logger
 - loopback address, 178
- M**
- ManagedConnectionFactory JavaBean, 350
 - master broker
 - management, 182-183
 - specifying, 173
 - maxBytesPerMsg destination property, 333, 369
 - maxNumActiveConsumers destination property, 335, 369
 - maxNumBackupConsumers destination
 - property, 335, 369
 - maxNumMsgs destination property, 333, 369
 - maxNumProducers destination property, 334, 369
 - maxTotalMsgBytes destination property, 334, 369
 - MBean server, introduced, 400
 - MBeans
 - server
 - See* MBean server
 - MDBs, *See* message-driven beans
 - memory management, for broker, 120
 - message-driven beans
 - Resource Adapter configuration for, 347, 351
 - message expiration, clock synchronization and, 67
 - message flow controls
 - broker, 120
 - connection factory attributes, 197
 - connection flow limits, 248
 - connection flow metering, 246
 - consumer flow limits, 247-248
 - tuning for performance, 246-248
 - message header overrides, 198-199
 - message service architecture, 241
 - message service performance, 238-242
 - messages
 - body type and performance, 238
 - broker limits on, 90, 120, 302
 - destination limits on, 304, 334
 - flow control
 - See* message flow controls
 - fragmentation, 127
 - metrics messages
 - See* metrics messages
 - pausing flow of, 110
 - persistence of, 125
 - purging from a physical destination, 290
 - reclamation of expired, 120, 303
 - size, and performance, 237-238
 - messageSelector ActivationSpec property, 351
 - metric information
 - brokers, 92
 - connection services, 100
 - physical destinations, 115

metrics

- about, 209

data

- See metrics data

messages

- See metrics messages

- topic destinations, 225

metrics data

broker

- See broker metrics

physical destination

- See physical destination metrics

- using broker log files, 217

- using message-based monitoring API, 225-226

metrics messages

- about, 224

- type, 225

metrics monitoring tools

- compared, 210-212

- message-based monitoring API, 224-227

- Message Queue Command Utility
(imqcmd), 218-222

- Message Queue log files, 217

- monitoring, *See* performance monitoring

- monitoring, support for Java ES, 223-224

- monitoring services, broker, 78, 209-210

- multiple queue consumers, 248

N

- NORMAL service type, 94

- nsswitch.conf file (Linux), 178

O

- object stores, 189-192

- file-system, 191-192

- file-system store attributes, 191-192

- initial context, 190

- LDAP server, 189-191

- LDAP server attributes, 190

- location, 190

- locations, 374, 375, 376

- object stores (*Continued*)

- security, 190

- operating system

- performance effect of, 239

- tuning Solaris performance, 242

- Oracle, 132

- overrides

- for message header, 198-199

- on command line, 75

P

- password file

- broker configuration properties, 137, 313

- command line option, 280

- location, 169, 374, 375, 377

- permissions, 168

- using, 168-169

- password managed connection factory attribute, 350

- password Resource Adapter attribute, 348

- passwords

- administrator, 144, 169

- default, 341

- encoding of, 311

- format, 142

- JDBC, 169

- LDAP, 169

- password file

- See password file

- SSL key store, 169

- SSL keystore, 162

- pausing

- brokers, 89, 286

- connection services, 97, 288

- physical destinations, 110, 290

- performance

- about, 229-232

- baseline patterns, 231-232

- benchmarks, 230-231

- bottlenecks, 233

- factors affecting

- See performance factors

- indicators, 230

- measures of, 230

- performance (*Continued*)
 - monitoring
 - See performance monitoring
 - optimizing
 - See performance tuning
 - reliability tradeoffs, 234
 - troubleshooting, 251
 - tuning
 - See performance tuning
- performance factors
 - acknowledgment mode, 236
 - broker limit behaviors, 241
 - connections, 239-240
 - data store, 241-242
 - delivery mode, 235
 - durable subscriptions, 237
 - file synchronization, 309
 - hardware, 239
 - JVM, 239
 - message body type, 238
 - message flow control, 246
 - message service architecture, 241
 - message size, 237-238
 - operating system, 239
 - selectors, 237
 - transactions, 235-236
 - transport protocols, 240
- performance monitoring
 - JES Monitoring Framework (JESMF)
 - See Java ES Monitoring Framework
 - metrics data
 - See metrics data
 - tools
 - See metrics monitoring tools
- performance tuning
 - broker adjustments, 245-246
 - client runtime adjustments, 246-248
 - process overview, 229-230
 - system adjustments, 242-245
- permissions
 - access control file, 138
 - admin service, 138
 - computing, 156-157
 - data store, 126
- permissions (*Continued*)
 - embedded database, 130
 - password file, 168
 - user repository, 141, 295
- persistence
 - about, 125
 - data store
 - See data store
 - file-based data store, 126-127
 - JDBC-based
 - See JDBC-based persistence
 - options (figure), 125
 - properties, 308-309
 - security for, 128, 132
- persistence services, broker, 77, 125-126
- physical destinations
 - auto-created
 - See auto-created destinations
 - batching messages for delivery, 306, 335
 - commands for managing, 289-291
 - compacting, 117
 - compacting file-based data store, 290
 - creating, 107, 289
 - destroying, 110, 289
 - disk utilization, 116-118
 - displaying metrics, 291
 - getting information about, 290
 - limit behaviors, 334
 - listing, 112, 290
 - managing, 105-119
 - metrics
 - See destination metrics
 - naming conventions, 107
 - pausing, 110, 290
 - properties of, 333-336
 - purging, 111-112
 - purging messages from, 290
 - querying, 113
 - restricted scope in cluster, 306, 335
 - resuming, 111, 290
 - temporary, 113
 - types, 289
 - updating attributes, 290
 - updating properties, 112

- physical destinations (*Continued*)
 - using dead message queue, 118
 - viewing information about, 112-116
 - viewing metric information, 115
- Port Mapper
 - about, 95
 - port assignment for, 278
- precedence (of configuration properties), 79
- producers
 - destination limits on, 305, 334
- production environment
 - administration tasks, 34-35
 - maintaining, 35
 - setting up, 34-35
- properties
 - auto-create, 303-307
 - broker instance configuration, 79
 - broker monitoring service, 317-321
 - cluster configuration, 321-324
 - connection services, 300-302
 - httpjms connection service, 394
 - httpsjms connection service, 394
 - JDBC-related, 309-310
 - Logger, 317-321
 - persistence, 308-309
 - physical destinations
 - See* physical destinations, properties of
 - routine services, 302-303
 - security, 311-313
 - syntax, 80
- protocol types
 - HTTP, 95
 - HTTPS, 95
 - TCP, 94, 95
 - TLS, 94, 95
- protocols, *See* transport protocols
- purging physical destinations, 111-112

Q

- querying
 - broker clusters, 176-177
 - brokers, 91
 - connection services, 99

- querying (*Continued*)
 - physical destinations, 113
- queue load-balanced delivery
 - auto-created queue, 305
 - auto-created queues, 305
 - behavior, 248
 - properties, 335
 - tuning for performance, 249
- queues, auto-created, 303
- quiescing brokers, 88, 286

R

- reconnectAttempts managed connection factory
 - attribute, 350
- reconnectAttempts Resource Adapter attribute, 349
- reconnectEnabled managed connection factory
 - attribute, 350
- reconnectEnabled Resource Adapter attribute, 348
- reconnectInterval managed connection factory
 - attribute, 351
- reconnectInterval Resource Adapter attribute, 349
- reconnection, automatic, *See* automatic reconnection
- reliable delivery
 - and flow control, 197
 - performance tradeoffs, 234
- reloadXMLSchemaOnFailure destination
 - property, 336
- Resource Adapter
 - properties, 347
 - reconnection, 348, 349, 350
- ResourceAdapter JavaBean, 347
- RESTART property, 70, 71
- restarting brokers, 88, 286
- resuming
 - brokers, 89, 286
 - connection services, 98, 288
 - physical destinations, 111, 290
- routine services, properties, 302-303
- routing services, broker, 77, 119

S

Secure Socket Layer standard, *See* SSL

security

authentication

See authentication

authorization

See authorization

encryption

See encryption

manager

See security manager

object store, for, 190

security manager

about, 135

properties, 311-313

security services, broker, 77, 135-138

selectors

about, 237

performance effect of, 237

self-signed certificates, 159-165, 385

sendUndeliverableMsgsToDMQ ActivationSpec property, 353

server, MBean, *See* MBean server

service (Windows)

Java runtime for, 72-73

reconfiguring, 72

removing broker, 73

running broker as, 71-74

startup parameters for, 73

troubleshooting startup, 73

service types

ADMIN, 94

NORMAL, 94

shutting down brokers, 87, 286

as Windows service, 73

Simple Network Time Protocol (SNTP), 67

SSL

about, 138

broker clusters, 180

connection services

See SSL-based connection services

SSL, enabling, 162

SSL

encryption and, 159

SSL-based connection services, implementing, 159

SSL-based connection services, starting up, 163

ssladmin connection service, 95, 159

ssljms connection service, 94, 159

starting

clients, 74-75

SSL-based connection services, 163

startup parameters for broker Windows service, 73

subscriptionDurability activation specification

attribute, 351

subscriptionDurability ActivationSpec property, 352

subscriptionName activation specification

attribute, 352

subscriptionName ActivationSpec property, 351

Sun Cluster, configuration for, 309

synchronizing

clocks, 67-68

memory to disk, 127

syntax for all commands, 277-278

syslog, 213, 216

system clock synchronization, 67-68

T

TCP protocol type, 94, 95

temporary physical destinations, 113

thread pool management

about, 96

dedicated threads, 96

shared threads, 96

time synchronization service, 67

time-to-live, *See* message expiration

TLS protocol type, 94, 95

tools, administration, *See* administration tools

topics, auto-created, 303

transactions

commands for managing, 291-292

displaying information about, 292

managing, 122-124

performance effect of, 235-236

transport protocols

performance effect of, 240

protocol types

See protocol types

transport protocols (*Continued*)

relative speeds, 240

tuning for performance, 243-245

troubleshooting, 251

Windows service startup, 73

tunnel servlet connection, 398

tutorial, 39

U

ulimit command, 68

unquiescing brokers, 88, 286

updating

brokers, 89-90

connection services, 98, 288

usage help, 86, 142, 285

useDMQ destination property, 335, 369

useDMQ property, 118

user data, 136

user groups

default, 138

deleting assignment, 139

predefined, 140

user names, 341

default, 141

format, 142

user repository

about, 136

activating and deactivating users, 144

changing passwords, 143-144

deleting users, 143

flat-file, 139-145

initial entries, 140

LDAP, 145-147

location, 374, 375, 377

platform dependence, 295

populating, 142-143

user groups, 139-141

userName managed connection factory attribute, 350

userName Resource Adapter attribute, 348

V

validateXMLSchemaEnabled destination

property, 336

version, displaying, 85, 142, 285

W

W32Time service, 68

wildcards

destination names, 107

publishers, 109

subscribers, 109

Windows service, *See* service (Windows)

write operations (for file based store), 127

X

XMLSchemaURIList destination property, 336

xntpd daemon, 67